NPS EC-93-014

NAVAL POSTGRADUATE SCHOOL

Monterey, California

ACOUSTIC PROPAGATION MODELING USING MATLAB

John P. Powers

1 September 1993

Prepared for:
Naval Postgraduate School
Monterey CA 93943

NAVAL POSTGRADUATE SCHOOL
Monterey, CA

Rear Admiral T.A. Mercer
Superintendent

Dr. Harrison Shull
Provost

This report was prepared as part of a research project entitled "Transient-wave Acoustical Imaging" sponsored by the Direct Funded Research Program of the Naval Postgraduate School, Monterey, California.

This report was prepared by:

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	1 September 1993	Interim Report 1 October 1991 - 1 July 1993

4. TITLE AND SUBTITLE

Acoustic Propagation Modeling Using MATLAB

5. FUNDING NUMBERS

6. AUTHOR(S)

John P. Powers

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Postgraduate School
Monterey CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

NPSEC-93-014

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Naval Postgraduate School
Reasearch Office (Code 08)
Naval Postgraduate School
Monterey CA 93943-5000

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

The views expressed in this report are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release;
distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This report presents a computational technique for the rapid, efficient calculation of fields from transient acoustic sources in linear, isotropic media. The source velocity is separable in space and time. The method uses a spatial impulse response method based on linear systems concepts to express the output in terms of the Green's function of propagation equation and the boundary conditions. The output is expressed as the inverse spatial transform of the product of the transform of the spatial excitation and a time-varying spatial filter that represents propagation. The calculation technique has been implemented in MATLAB and sample cases are presented for the circular and square piston, as well as a Gaussian- and Bessel-weighted spatial excitation. Code for the MATLAB implementation is provided.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Acoustic propagation, transient waves, transfer function, linear systems theory	64
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAR

Abstract

This report presents a computational technique for the rapid, efficient calculation of fields from transient acoustic sources in linear, isotropic media. The source velocity is separable in space and time. The method uses a spatial impulse response method based on linear systems concepts to express the output in terms of the Green's function of propagation equation and the boundary conditions. The output is expressed as the inverse spatial transform of the product of the transform of the spatial excitation and a time–varying spatial filter that represents propagation. The calculation technique has been implemented in MATLAB and sample cases are presented for the circular and square piston, as well as a Gaussian– and Bessel–weighted spatial excitation. Code for the MATLAB implementation is provided.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

The propagation characteristics of continuously radiated monochromatic ultrasonic sources are well solved through application of the angular spectrum technique [1] or Fresnel integrals. More frequently, however, acoustic imaging, tissue characterization, and physical acoustics applications tend to use pulsed sound. The propagation of pulsed ultrasound with arbitrary temporal and spatial components is not understood to the same degree. We want to develop a reliable, easy method of diffraction prediction. This report describes an approach based on linear systems theory and the Fourier transform. The goal was to achieve a readily usable method of predicting pulsed wave diffraction in a time–efficient and accurate manner in order to examine the wave diffraction. Earlier efforts had used FORTRAN code to implement the propagation model on both a mainframe computer [2]– [6] and a personal computer [7]. Merrill had attempted to use MATLAB (a commercial matrix manipulation software package) to perform the calculations on a PC but was thwarted by memory restrictions of the earlier versions of this software. More recent versions allow larger matrices and arrays to be manipulated (subject only to the computer memory available) Thesis work performed by Upton [8] and Reid [9] applied this updated software to the problems of optical propagation and acoustic propagation, respectively. Much of the foundation for the work reported here was done by these officer–students.

The basic method of the spatial impulse response was introduced by Stepanishen [10]–[13], reviewed by Harris [14], and modified by Guyomar and Powers [5]. The approach of Guyomar and Powers differed by the use of linear systems theory. Linear systems theory revealed the importance of the total impulse response and its equivalence to the Green's function. Furthermore, the spatial impulse response functions are found in the spatial transform (Fourier) domain. Working in the transform domain allowed propagation of the wave to be viewed as a time–varying spatial filter applied to the spatial spectrum of the input excitation. The advantage of this method is that it provides the diffracted field from an insonifying wave with arbitrary temporal and spatial dependence in a computationally

efficient form. By use of the Fourier transform, an efficient computer implementation of this technique using FFT routines is possible.

The desired benefit of a fast, time–efficient computer implementation to calculate the acoustic potential or pressure is to aid in ultrasonic transducer design for medical, acoustic imaging, and mine warfare applications. With a knowledge of wave diffraction phenomenon a diffracted wave reflected from an unknown object can be used to provide information about the object. This type of system must be portable as well as time–efficient, which is very achievable given the trends in computer technology. Computers have become faster and have increased memory capacity while their size has decreased. Other benefits are derived from the use of the matrix manipulation program, MATLAB, and the ability to expand this implementation to cases involving lossy media. Because MATLAB is readily available on the commercial market, it requires no special equipment for computer implementation.

1.2 Report Overview

Chapter II consists of the problem description, including the source–to–observation plane geometry, a discussion on the linear systems approach, the mathematical development, and an overview of MATLAB and AXUM. Chapter III consists of a discussion of the two program modules, the acoustic filter module, AC_FIL.M, and the acoustic propagation module, AC_PROP.M. Chapter IV starts with the set of defining parameters. These parameters are then used for an explanation of the program's verification and an investigation of other input excitation functions. Chapter V contains some examples of the numerical output for various source excitation conditions.

Following a summary in Chapter VI, Appendices A and C give detailed explanations of the MATLAB routines that were used to produce the numerical results, AC_FIL.M and AC_PROP.M. Appendix B gives examples of the propagation filters generated by the code in Appendix A. The source code of the various geometric excitation functions is given in Appendix D. Representative output presentations using a scientific visualization program, called Spyglass Dicer, are found in Appendix E.

Chapter 2

Problem Description

Before assembling a computer implementation, we must first understand the problem. An explanation of the problem follows, beginning with the description of the geometry in the first section. Section two continues the explanation into the linear systems approach. The third section proceeds through the mathematical development of the problem and ties in the Fourier transform. The theory presented in the first three sections was derived from the works of Guyomar and Powers [3, 4, 5, 6]. The final section gives an overview of the software tools used for generating the excitation functions, the propagation fields, and the output graphics.

2.1 Geometry

The problem geometry is shown in Fig. 2.1. The acoustic velocity potential $\phi(x, y, z)$ is to be calculated at an arbitrary point in the positive–z half–space given the z–directed velocity in a portion of the source plane. The source's z–directed velocity is spatially and temporally arbitrary and is rigidly baffled (i.e., equal to zero) in the region outside the source. Furthermore, it is assumed that the spatial and temporal components of the z–velocity are separable, having the form at the input plane

$$v_z(x_0, y_0, 0, t) \;=\; T(t)s(x_0, y_0)\,. \tag{2.1}$$

A linear, homogeneous, and lossless medium is assumed to be present between the source and observation point. (The extension to lossy propagation can be found in Ref. [15].)

2.2 Linear Systems Approach

Linear systems solutions are applied to systems that are linear and time–invariant. A linear systems solution approach to this problem is possible because propagation in a linear homogeneous media is a linear, space–invariant process [3]. In linear systems the *impulse response* is the response of the system to an impulsive input. The *total impulse response*

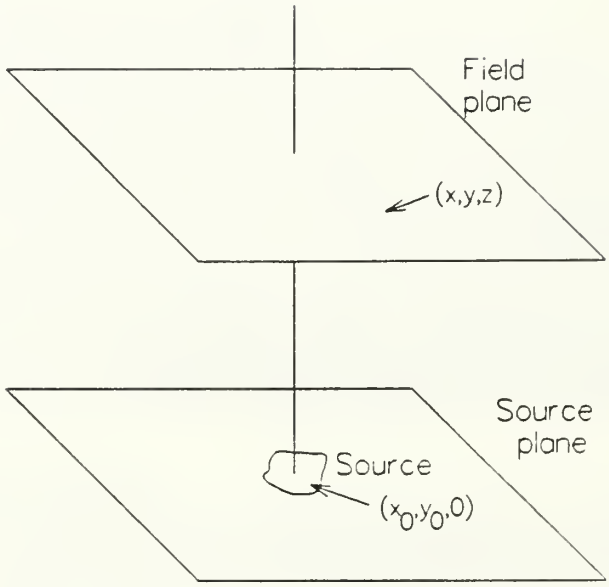


Figure 2.1: Source–to–receiver geometry.

$h(x,y,z,t)$ of a system is produced by an input of the form $\delta(x,y,t) = \delta(x,y)\delta(t)$; this is shown in Fig. 2.2a. Figure 2.2b shows the *spatial impulse response* $p(x,y,z,t)$, defined as the response to an excitation of the form $s(x,y)\delta(t)$. Note that an arbitrary spatial input has been substituted for the impulsive spatial input. Recall from linear systems theory that the solution for an arbitrary input (spatial or temporal) is the convolution of the input with the system's total impulse response; therefore, the spatial impulse response in this case is given by

$$p(x,y,z,t) \;=\; s(x,y) \overset{*}{}_x \overset{*}{}_y h(x,y,z,t)\,, \tag{2.2}$$

where $\overset{*}{}_x$ indicates convolution with respect to the variable shown.

The double convolution in Eq. 2.2 is not easily computer implemented. The Fourier transform furnishes a convenient method to resolve this dilemma by using the property that convolution in the spatial domain becomes multiplication in the transform domain, i.e.,

$$\tilde{p}(f_x,f_y,z,t) \;=\; \tilde{s}(f_x,f_y)\tilde{h}(f_x,f_y,z,t)\,, \tag{2.3}$$

where the tilde notation indicates the Fourier transform of the function (in this case, the two–dimensional spatial Fourier transform). This relation is shown in Fig. 2.2c.

The general solution is shown in Fig. 2.2d, where

$$\phi(x,y,z,t) \;=\; s(x,y)T(t) \overset{*}{}_x \overset{*}{}_y \overset{*}{}_t h(x,y,z,t)\,. \tag{2.4}$$

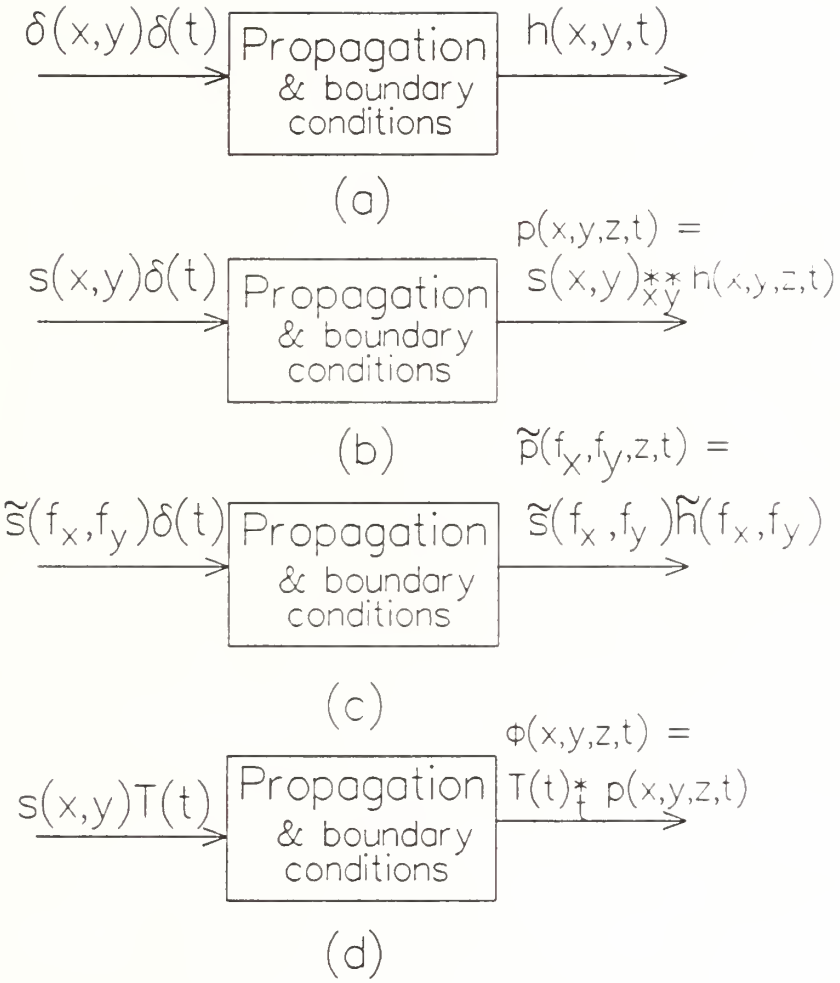


Figure 2.2: Block diagram of (a) the total impulse response $h(x,y,t)$), (b) the spatial impulse response $p(x,y,z,t)$ in the space–time domain, (c) the spatial impulse response $\tilde{p}(f_x,f_y,t)$ in the spatial frequency–time domain, and (d) the general solution $\phi(x,y,z,t)$.

Substituting Eq. 2.2 into Eq. 2.4 gives

$$\phi(x,y,z,t) \;=\; T(t) \overset{*}{_t} p(x,y,z,t)\,. \tag{2.5}$$

It follows from Fig. 2.2c and Eqs. 2.2 and 2.5 that the key to finding the general solution is finding the spatial impulse response $p(x,y,z,t)$ which is, in turn, dependent on the *total impulse response* of the system $h(x,y,z,t)$. This total impulse response of the system is the propagation field that results from an impulsive source, as in Fig 2.2a, that solves the wave equation and satisfies the boundary conditions. The solution to the wave equation satisfying the boundary conditions is commonly known as the *Green's function*; hence, the total impulse response is simply the Green's function. Therefore, once the Green's function is known, the total impulse response is also known, and the solution becomes a triple convolution between an excitation source which is spatially and temporally arbitrary (and assumed to be known), and the systems' total impulse response or Green's function. The two spatial convolutions are easily implemented in the spatial transform domain; the time convolution can be implemented in the temporal transform domain, if desired. We have found it easier to perform the time convolution directly in the time domain.

2.3 Mathematical Development for Acoustic Propagation

The wave equation for lossless media is

$$\nabla^2\phi - \frac{1}{c^2}\frac{\partial^2\phi}{\partial t^2} \;=\; 0\,. \tag{2.6}$$

The general solution of Fig. 2.2c gives the result in terms of the acoustic potential ϕ which must be found from a z-velocity input. We relate the acoustic velocity and acoustic potential with the following relationship;

$$v(x,y,z,t) \;=\; -\nabla\phi(x,y,z,t)\,. \tag{2.7}$$

Hence, the z-velocity component is given by

$$v_z(x,y,z,t) \;=\; -\frac{\partial\phi(x,y,z,t)}{\partial z}\,. \tag{2.8}$$

Since the wave equation is in terms of c (the acoustic velocity in the media) and time t, the partial derivative with respect to z must be related to these two parameters. This is done by using the fact that a wave traveling in the positive-z direction has an argument of the form $ct - z$, resulting in the relationship

$$\frac{\partial}{\partial z} \;=\; -c\frac{\partial}{\partial t}\,. \tag{2.9}$$

Applying Eq. 2.9 to Eq. 2.8 gives the z-velocity at the input plane ($z = 0$) as

$$v_z(x,y,z,t) \;=\; c\frac{\partial\phi(x,y,0,t)}{\partial t}\,. \tag{2.10}$$

Equation 2.10 requires the acoustic potential at the input plane. It was assumed in Eq. 2.1 that the z-velocity is separable which also implies a separable acoustic potential. Such an acoustic potential has the general form

$$\phi(x,y,0,t) = s(x,y)\tau(t) \tag{2.11}$$

at the $z = 0$ plane. If Eq. 2.11 is substituted into Eq. 2.10 and the partial derivative is carried out, then Eq. 2.10 becomes

$$v_z(x,y,z,t) = c\, s(x,y)\tau'(t)\,, \tag{2.12}$$

where the prime indicates a derivative with respect to the time variable t. A comparison of the z-velocities given in Eqs. 2.1 and 2.12 indicates that $T(t)$ is equivalent to $c\,\tau'(t)$. Equation 2.12 is now the input in Fig. 2.2c.

As stated earlier, the general solution of Fig. 2.2c is the Green's function. For the standard wave equation (for lossless media) and rigidly baffled boundary conditions, the applicable Green's function is [4]

$$h(x,y,z,t) \;=\; \frac{\delta(ct-z)}{2\pi R} \tag{2.13}$$

where $R = \sqrt{x^2+y^2+z^2}$. Substituting this into Eq. 2.2 provides the spatial impulse response

$$\begin{aligned} p(x,y,z,t) &= s(x,y) \overset{*}{}_x \overset{*}{}_y h(x,y,z,t) \\ &= s(x,y) \overset{*}{}_x \overset{*}{}_y \frac{\delta(ct-z)}{2\pi R}\,. \end{aligned} \tag{2.14}$$

Thus, the spatial impulse response in the spatial transfer domain is the product of the angular spectrum of the source $\tilde{s}$ and the propagation transfer function $\tilde{p}$, written symbolically as

$$\tilde{p}(f_x,f_y,z,t) \;=\; \tilde{s}(f_x,f_y,t)\tilde{h}(f_x,f_y,z,t)\,. \tag{2.15}$$

Taking the two–dimensional spatial Fourier transform of the Green's function in Eq. 2.13 gives the propagation transfer function

$$\begin{aligned} \tilde{h}(f_x,f_y,z,t) &= \mathcal{F}\left\{\frac{\delta(ct-z)}{2\pi R}\right\} \\ &= 2J_0\left(\rho\sqrt{c^2t^2-z^2}\right) H(ct-z)\,, \end{aligned} \tag{2.16}$$

where the relationship

$$\mathcal{F}\left\{\frac{\delta\left(t-\frac{z}{c}\right)}{R^n}\right\} \;=\; \frac{J_0\left(\rho\sqrt{c^2t^2-z^2}\right) H(ct-z)}{(ct)^{n-1}} \tag{2.17}$$

was applied (with $r = \sqrt{f_x^2 + f_y^2}$). The term $H(ct - z)$ is the Heaviside step function and makes the wave causal.

Equation 2.16 exhibits two important points. The first is that the propagation transfer function is a Bessel of the first kind of zero order. Secondly, the propagation transfer function can be identified as a time–varying spatial filter having a Bessel shape. The filter begins as an all–pass filter early in its time evolution and then becomes more of a low–pass filter as time progresses. This increasingly narrow low–pass filter reduces the resolution in the receiving plane of the system as time progresses. In our computer programs, the propagation spatial filter at various instants of time are produced by the MATLAB file, AC_FIL.M.

Equation 2.16 is valid for an input that is temporally impulsive and spatially arbitrary. Taking the inverse two–dimensional spatial transform of Eq. 2.16 would, in this case, give a final result since convolution with an impulse is the same function at the time that the impulse occurred. To account for a non–impulsive time input component, the convolution of Eq. 2.5 must be carried out, resulting in

$$\phi(x, y, z, t) \quad = \quad T(t) \overset{*}{_t} \mathcal{F}^{-1} \{\tilde{p}(f_x, f_y, z, t)\} \ , \tag{2.18}$$

where $\tilde{p}$ is the product of the angular spectrum of the source $\tilde{s}$ and the propagation transfer function $\tilde{h}$. Since only temporally impulsive inputs are simulated in the examples that follow, the convolution in Eq. 2.18 was not computed for our cases. Our final solution, therefore, reduces to taking the inverse 2–D spatial Fourier transform of the spatial impulse response. The program module, AC_PROP.M, simulates the spatial impulse response solution for the input excitation function distribution chosen by the user.

The program that implements these equations is discussed in Chapter III and detailed in Appendix C. Illustrative examples of the time–varying filters and outputs follow in the Chapter IV discussion with more examples supplied in Appendices B and E. The tool of implementation for the program was MATLAB with graphical assistance provided by AXUM. Output data was also represented using a scientific visualization program called Spyglass Dicer. An overview of these programs follows in the next section.

2.4 Software Tools

2.4.1 MATLAB Overview

MATLAB is a high–performance, interactive numeric computation software package for science and engineering applications produced by The MATHWORKS, Inc. The name comes from MATrix LABoratory; hence, the basic data element is a matrix (or array), which does not require dimensioning. MATLAB. A major advantage of MATLAB is that it uses a "vectorized" approach to computations, simplifying the programming. Another distinct advantage is the availability of preprogrammed functions, such as calculation of the two–dimensional FFTs and the Bessel function [16].

In MATLAB there are two type of macro–like files. A *script file* is used to automate long sequences of commands including functions. Arguments are not passed into script files. A *function file*, however, may have arguments passed into it and out of it. Examples of script files in this thesis include AC_FIL.M and AC_PROP.M. Examples of functions include the input functions, the three dimensional graphing function mesh, and the two "fft" functions that realize the Fourier transform [16].

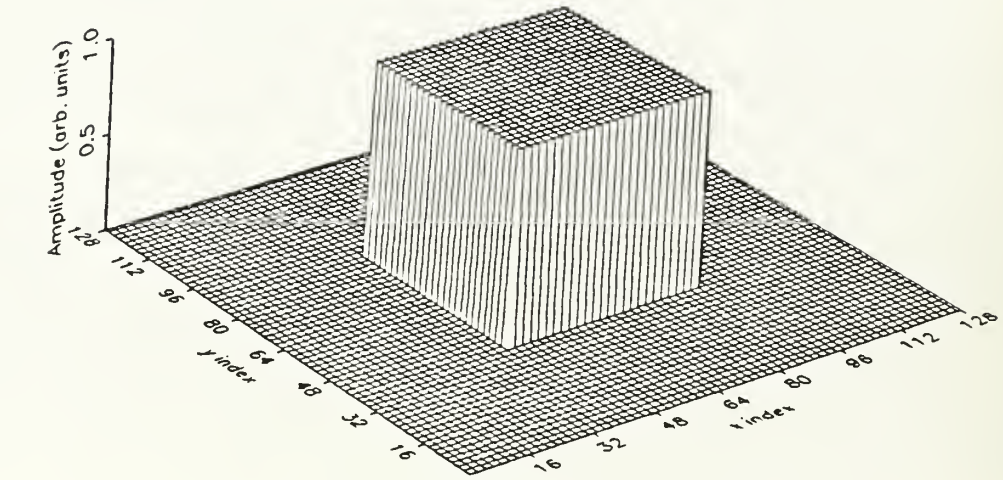
The two "fft" functions employed for the Fourier transform are **fft2** and **fftshift**. The **fft2** function is a two–dimensional fast Fourier transform and is repeatedly used throughout our program. MATLAB allows the use of only positive indices in an array. Geometrically, this is equivalent to working only in the first quadrant of an $x - -y$ plane. Our sources are symmetric around the origin and we want to display them using all four quadrants. The key to the transformation is the **fftshift** function. Figure 2.3a. shows the first–quadrant representation of a rectangular source centered on the origin. (Note the MATLAB assumes implicitly that the function is periodic in the x and y directions. Periodic translation of the first–quadrant in the x and y directions will fill in the rest of the source at the origin.) The **fftshift** function rearranges the data so that it is centered in the first quadrant. (Actually the data is not quite centered when one uses an even number of sample points, since there are not any sample points aligned with the center. The shifted array is offset from the quadrant center by one–half of a sample interval in both the x and y directions.) Figure 2.3b represents the shifted source. For viewing purposes, the shifted version is easier to understand; for computational purposes the unshifted function must be used to avoid phase errors when calculating the transform or its inverse. We will use the terms "shifted" for functions represented in the centered geometry and "unshifted" for the functions represented in the corner geometry. The interested reader is referred the MATLAB User's Guide [16] for more details.

The MATLAB **mesh** command can be used to provide three–dimensional plots of the computed fields. Alternatively, the input and output data can be stored in ASCII format, imported into a more powerful graphics package, called AXUM, and plotted.

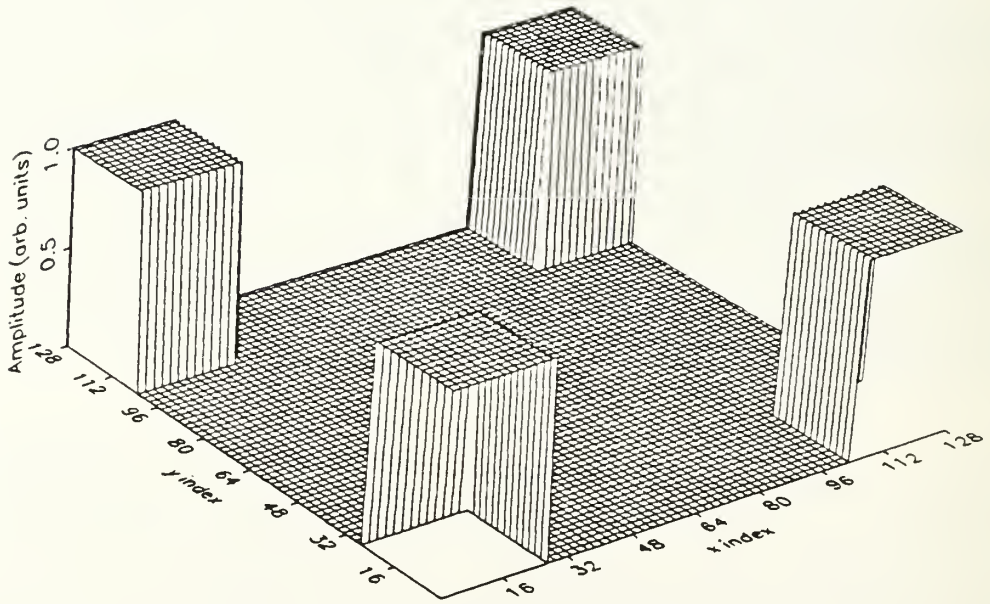
2.4.2 AXUM Overview

AXUM is an interactive software package for technical graphics and data analysis, produced by TriMetrix Inc. AXUM allows easy manipulation of raw data imported from MATLAB in ASCII format. (Data may also be imported from several other formats.) Once imported, data manipulation can be carried out by AXUM through use of the *Transform* and *Convert* menu options. Then the data is arranged using the *MAT2GRID* routine found in the *History Editor* menu, so that the desired surface plot can be generated. The *MAT2GRID* routine produces three columns of data from the original data array by placing each element of acoustic potential data in the z data column with the indices from the array in the x and y columns. Once processed, the data is ready for graphing [17].

The *Graph* menu gives various options for controlling the graph's attributes and general aesthetics. Once the desired axes intervals, labels, and titles have been set, the graph can be displayed in the *Edit Screen* window. The *Edit Screen* window allows changes to be



(a)



(b)

Figure 2.3: Illustration of the center versus corner geometry.

made to the graph while it is displayed. (This can be is time–consuming because the graph is redrawn after each change.) A useful feature of the *Edit Screen* window, however, is the capability to rotate and tilt the graph interactively so that the preferred aspect is achieved. After establishing the desired attributes, the graph can be saved as a graph or as an image. The concept of saving the graphs as images is another very useful feature because it allows a single graph to be assembled and saved as a graph template on which the other images may be overlaid.

2.4.3 Spyglass Software Overview

The Spyglass software consists of a set of commercial programs produced by Spyglass, Inc. for the purpose of aiding in visualization of scientific data on Macintosh computers. (One program, the Transform program, is also commercially available for Unix workstations.)

The Spyglass Data Utility has the capability to read data from a wide variety of formats into the HDF (Hierarchical Data Format) format [1] used by the Spyglass software. In particular, for our work, the utility is able to read ASCII data. In our case, we took the 64 ASCII data files produced by propagation model and transferred them to a Macintosh II computer. The Spyglass Data Utility was used to create an HDF file, representing a data volume of 128x128x64 samples. (The typical data compression was from 16 MB of ASCII data to a 6–MB HDF file.)

Spyglass Dicer is a program that allows the user to interactively look at the 3–D data volume in various ways. Planes of any arbitrary orientation can be inserted or rectangular regions of the data can be observed. (Figures 4.3 and 4.4 on pages 24 and 25 are representative of the data views that can be obtained with the Dicer program.)

The Spyglass Transform program allows the user to pick a plane of the HDF data volume (currently, the planes must be oriented perpendicular to one of the axes) and to view it in a variety of formats, such as a color intensity display, a gray–scale intensity display, or a surface plot.

[1] The HDF format is becoming a popular means of transferring large data files between computing platforms.

Chapter 3

MATLAB Modeling of Equations

This chapter discusses the two MATLAB files that implement the model, AC_FIL.M and AC_PROP.M. Together, these script files form the two modules of the program that simulate acoustic wave diffraction via implementation of the concepts and equations presented in the previous chapter. The program was written in two modules since the propagation filter characteristics of the medium depend only on the locations of the source and receiving plane locations and are independent of the source geometry. The propagation filters could be computed once and then be reused for a variety of excitation conditions. This modularity of approach was fortuitous, since the calculation of the Bessel functions in the filters was computationally intensive. (This modular approach to the calculation of the fields is one of the major advantages of our method.) A working narrative of AC_FIL.M opens the discussion, followed by a working narrative of AC_PROP.M. The excitation functions, or input functions, are included under the AC_PROP.M heading. A brief summary of the program steps follows in the final section.

3.1 Acoustic Filter Module

The script file, AC_FIL.M (ACoustic FILter) computes the time-varying Bessel filters of Eq. 2.16, repeated here for convenience as

$$\begin{aligned} \tilde{h}(f_x, f_y, z, t) &= \mathcal{F}\left\{\frac{\delta(ct-z)}{2\pi R}\right\} \\ &= 2J_0\left(\rho\sqrt{c^2t^2-z^2}\right) H(ct-z)\,. \end{aligned} \tag{3.1}$$

Before discussing the coding of Eq. 3.1, the array geometry must be explained.

The array geometry is set by the size of the array and the sampling frequency. The variable *base* denotes the number of points on a side in the base array creating an $base \times base$ array. (It is advantageous in calculating an FFT to make *base* a power of 2; this is not necessary, however.) Initially *base* was given a value of 64 as in previous work [3]— [7]. Once the program result was verified, *base* was increased by a factor of two to 128. Making

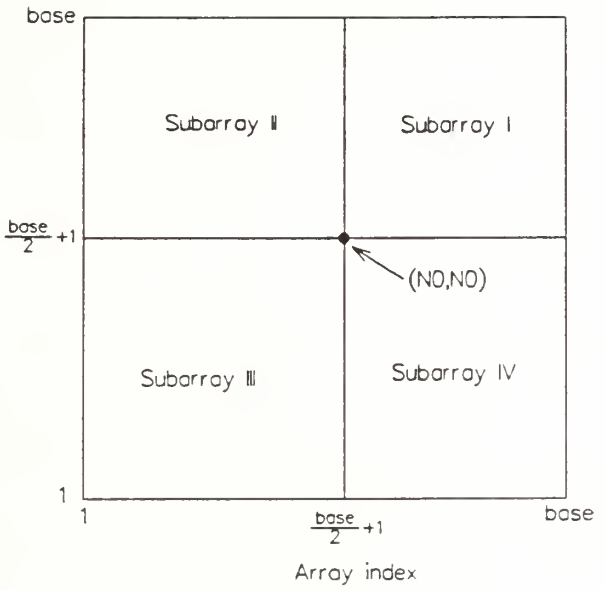


Figure 3.1: Offset geometry of base array matrix.

base an even number, however, causes the center of symmetry of the array to fall between array elements. The center of symmetry of the source $N0$ coincided with the array element at

$$N0 = \frac{base}{2} + 1 . \tag{3.2}$$

This results in an offset geometry as shown in Figure 3.1.

The offset center at $(N0, N0)$ divides the base array into four subarrays, each of different sizes. The fact that the subarrays have different sizes is important in the use of symmetry because only one subarray is actually computed; the other subarrays are determined by symmetry. We calculate the data for the $N0 \times (N0 - 1)$ points in Subarray I, shown in Fig. 3.1. In addition, data is calculated for an additional column $(N0 \times 1)$ located at the right side of Subarray I. (The extra column was required to compute all of the values in the other three quadrants using symmetry.) Subarrays II, III, and IV of Fig. 3.1 are determined from Quadrant I using symmetry with MATLAB's **flip** commands. The use of symmetry in this manner was employed in both program modules. It should be noted that this assumed x- and y-axis symmetry imposes an other restriction on the assumed source geometry; this restriction can be removed by performing the computations over the full dimensions of the array rather than relying on symmetry to simplify the calculations.

The number of time samples (or number of time slices), *slices*, was set at 64. Initially, this was to emulate previous work [3]— [7]; however, the time duration and resolution proved to be adequate for the follow–on simulations. Although there are 64 time slices,

only 61 filters are generated by the MATLAB implementation of Eq. 3.1. The Heaviside step function in Eq. 3.1 was simulated by arbitrarily setting a variable *Step* to three which produces three all–zero rows for the first three time slices. The result is that *slices* – *Step*, or 61, time slices actually require computing. Since the first three time slices are zero, the computations start with the fourth time slice at time z/c and proceed to the time value, *time_max*, provided by the user.

Referring back to Eq. 3.1, it is seen that the argument to the Bessel function, $\rho\sqrt{c^2t^2 - z^2}$, is composed of four variables. Of the four, only the time t varies within the program. In the MATLAB code, time t is represented by the variable *time*. As previously stated, filter generation does not start until time z/c and is linearly incremented to the maximum time of propagation *time_max*. The source–to–receiver distance z was assigned the value $z = 10$ cm in the MATLAB code and c, the acoustic velocity in the medium, was assigned the value $c = 1500$ m/s (velocity in water). The value of z was originally chosen to parallel the previous work [3]— [7] and was found to be convenient for subsequent simulations. The value of *time_max* was set to 150 μs for the verification phase and to 375 μs for the remainder of the simulations. Consequently, time ranged from 66.667 μs to 150 μs (or 375 μs) in 61 increments.

The final variable in the Bessel argument to be examined is ρ. In the MATLAB code, ρ was given the name *rho* and has a maximum value, *rho_max*, of 200. The value of *rho_max* = *200* was arbitrarily chosen subject to the constraint that it needs to be large enough that the field does not extend beyond the edges of the array at its largest width (or else the field will be aliased back into the array from the edges). The value of 200 was chosen following Merrill [7]. Although *rho* is not time–varying, it does vary with spatial frequency

$$\rho = \sqrt{f_x^2 + f_y^2}. \tag{3.3}$$

A vector having $N0 - 1$ points extending from 0 to *rho_max* was then formed. Then, the vector was used in the MATLAB routine **meshdom** [16] to form two identical matrices *rhox* and *rhoy*. The **meshdom** routine takes a given vector and forms two matrices with the same spacing in the x direction and y direction. The two matrices, *rhox* and *rhoy* represent f_x and f_y in Eq. 3.3 and Fig. 3.2, which shows graphically the construction of *rho*. In Fig. 3.2, the inner scales are in terms of the column and row numbers, respectively. The row index runs from 1 to *N0*; the column index runs from *N0* to *base*. The x and y axes can be rescaled to units of spatial frequency (f_x and f_y) with units of cycles/m. This is represented in Fig. 3.2 by the outer labeling.

The combination of *rho* and the other variables forms the argument *arg* to the MATLAB Bessel function. Since time varies for each time slice, *arg* varies for each time slice generating a filter per time slice. After generation each filter is stored to disk for use by AC_PROP.M. The variables *base*, *N0*, *slices*, and *Step* are also stored to disk for use by AC_PROP.M. The interested reader can find a detailed explanation and the source code in Appendix A. Graphical examples of the time–varying filters are include as Appendix B.

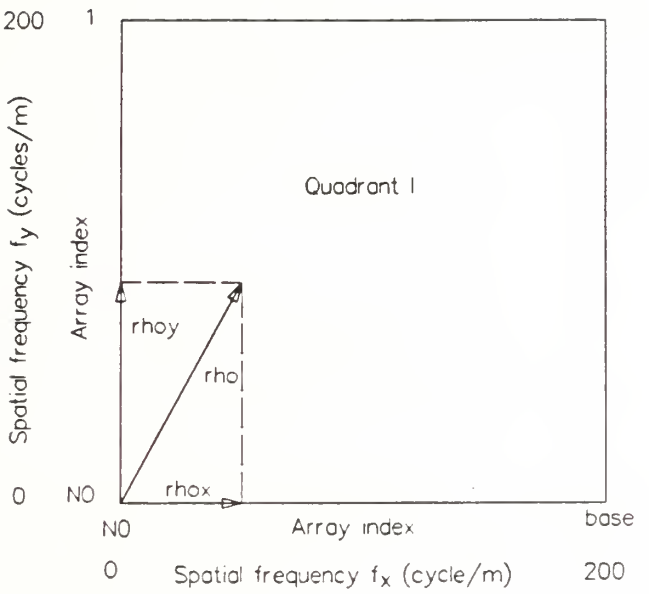


Figure 3.2: Construction of rho shown graphically.

3.2 Acoustic Propagation Module

The script file, AC_PROP.M (ACoustic PROPagation), allows the user to specify the type and dimensions of the input excitation. It then takes the user's chosen input function and simulates propagation as a function of time. AC_PROP computes the spatial impulse response $p(x, y, z, t)$,

$$p(x, y, z, t) \;=\; \mathcal{F}^{-1}_{f_x, f_y} \left\{ \tilde{s}(f_x, f_y) \tilde{h}(f_x, f_y, z, t) \right\} , \tag{3.4}$$

using the time–varying Bessel filters, produced by AC_FIL for $\tilde{h}(f_x, f_y, z, t)$. In Eq. 3.4, $\mathcal{F}^{-1}_{f_x, f_y} \{\cdot\}$ is the inverse spatial Fourier transform operator and the $\tilde{\text{ }}$ denotes a transformed function.

To compute $p(x, y, z, t)$, AC_PROP first loads the variables passed from AC_FIL and then queries the user to make an input function choice. The user is given four input functions from which to choose: *Circle*, *Table*, *Gaussian*, and *Bessel*. The *Table* and *Circle* are equal–amplitude sources having the shape of a square and a circle, respectively (known in acoustics as the square and circular piston). The Gaussian and Bessel inputs are circularly truncated functions that have the indicated amplitude distribution within the circle (i.e., the amplitudes are spatially varying across the face of the source). Pursuant to the input function choice, the user is asked to input the diameter d of the truncating circle (or width of the truncating square w in the case of the *Table* function). Once the program was verified, a diameter d of 51 was used; this value is available as the default in the menus. In the

case of the Gaussian or Bessel input, the user is further requested to input the standard deviation *sigma* or a scaling factor *a*, respectively. These two function defining parameters were varied on a case–by–case basis.

The variable *shft_input* holds the computed input array that was generated by the function written to model the chosen excitation function (Appendix D). (The reader is reminded that *shft* prefix in a variable name indicates the array has the centered geometry as discussed in Chapter II.) From *shft_input*, *F_input* is created by shifting (**fftshift**) *shft_input* to a corner geometry and then taking the two–dimensional spatial Fourier transform (**fft2**). As explained in Chapter II, the **fftshift** operation is necessary before the Fourier transform operation to obtain the correct phase relationship in the transform operation. With a shift back to the center geometry, the angular spectrum of the source $\tilde{s}(f_x, f_y)$, called *shft_F_input* in the program, is created for user viewing, if desired. The Bessel–function propagation transfer function $\tilde{h}(f_x, f_y, z, t)$ from Eq. 3.1 must now be loaded so that the angular spectrum–propagation transfer function product $\tilde{s}\tilde{h}$ on the right side of Eq. 3.4 can be formed. The loading and multiplication process is repetitive since *shft_F_input* must form a product with the filter (or appropriate propagation transfer function) from each time slice. This repetitive multiplication is accomplished with a loop.

The product of $\tilde{s}$ and $\tilde{h}$ (called *shft_F_output*). To find the desired result, the two–dimensional inverse spatial Fourier transform (**ifft2**) must be taken. Before this can be done, *F_output* is formed by shifting *shft_F_output* from the centered geometry to the corner geometry. Executing the inverse transform of the product yields the output (*output*) which is then shifted to give *shft_output* for viewing. The array *shft_output* represents the output at the time slice that the loop is currently computing; *shft_output* does not depict the acoustic potential (or propagation pattern) through time; it only depicts the acoustic potential at a specific time.

To produce a time history of the desired output, the center row (row *N0*) of *shft_output* is taken and placed in the array *output_plot* as the m-th column (m is the loop counter which relates directly to the time slice number, i.e., when $m = 4$, the computations for the fourth time slice are performed). This results in an array whose size is *base* × *slices* when *Step* zero–valued rows are added (that precede time slice *N0* row. The output examples are graphical interpretations of *output_plot*. Results generated in this manner are given in Chapter IV for all of the excitation functions. A detailed explanation of AC_PROP is provided in Appendix C.

3.3 Program Summary

The previous two sections gave an overview of the two program modules, AC_FIL and AC_PROP, including the code variable names and values assigned. What follows here is a summary of the steps that the program accomplishes. Step one is accomplished by AC_FIL. In this step the *slices-Step* filters to be used by AC_PROP are generated and saved.

AC_PROP generates the user–specified excitation function $s(x, y)$. Then the angular spectrum of the source $\tilde{s}(f_x, f_y)$ is computed by taking the 2-D spatial Fourier transform

of $s(x,y)$. The product $\tilde{s}\tilde{h}$ is computed for each time slice via a loop. In the loop, the inverse 2–D transform of the product forms an output for the specific time slice. The $N0$-th (center) row of that output is then placed in successive columns of a new array to form the final output, $p(x,0,10,t)$.

In the following chapter, examples of the final outputs are given. The excitations used for verification, as previously related, are the *Table* and *Circle* excitation functions. New values, as explain in this chapter, were then used for the simulation of the Gaussian and Bessel excitations.

Chapter 4

Numerical Simulations

In the previous chapter, a functional explanation of the two program modules was given including values assigned. The first section of this chapter reiterates the defining parameters, gives a brief explanation of each, and gives the parameter's units. In the following section, the defining parameters are given the values used to verify correct operation of the program for the circular and square piston sources. The last section presents results for the non-piston Gaussian and the Bessel excitation functions.

4.1 Defining Parameters

A *defining parameter* is a parameter that delineates an aspect of the basic setup upon which all the remaining parameters or variables depend. In the work of this thesis there are two sets of defining parameters—those for the filter generation and those for generation of the excitation functions. The filter parameters are found at the beginning of AC_FIL. AC_PROP reads these parameters from a data file that is stored with the results of the filter calculations.

The defining parameters found in AC_FIL include *base*, *slices*, *Step*, *c*, *z*, *time_max*, and *rho_max*. The first parameter *base* sets the dimensions of the base array, giving the number of sample points. The dimensions of the base array are, therefore, *base* x *base* where *base* is required to be a power of two (typically, 128). Making *base* a power of two allows MATLAB to use a high-speed radix-2 fast Fourier transform algorithm [16] to compute the spatial transforms. The next parameter *slices* is the number of time samples. Of these *slices* time slices, *slices-Step* slices require filters to be computed; the parameter *Step* is the number of leading-zero rows in the *base* x *slices* output array; as explained in the preceding chapter, this simulates the Heaviside step function. The parameters *base*, *slices*, and *Step* are unitless and are stored by AC_FIL in a file for use by AC_PROP.

The remaining defining parameters of AC_FIL have units and are used only in the computations of the filters. The acoustic velocity in the medium, free-space in this case, is denoted by the parameter *c* having the units of $\mathrm{m{\cdot}s^{-1}}$. The source-to-reception point distance has the designation *z* with units of meters. The maximum time of propagation

Parameter	Definition (units)
*base**	Size of square base array
*slices**	Number of time slices
*Step**	Number of leading zero–rows
*c**	Acoustic velocity in media (m/s)
*z**	Distance, source–to–receiver (m)
*time_max**	Maximum time of propagation (s)
*rho_max**	Spatial radius of the filters (length^{-1})
* passed to AC_PROP.M	

Table 4.1: Summary of the defining parameters in AC_FIL.

time_max has units of seconds. The spatial–frequency radius of the filters *rho_max* (or *rho*) has units of inverse length (i.e., m^{-1}, cm^{-1}, etc.). The unit of length depends on the area to be represented by the base array. These four parameters relate directly to Eq. 3.4 and are the parameters that dictate the diffraction properties of the filters. Table 4.1 summarizes the defining parameters found in AC_FIL.

Another important set of defining parameters is the set that defines the user chosen input. These input defining parameters are entered by the user when requested by AC_PROP. Once the input function is chosen, the diameter of the truncation circle d is input. (The width of the table w (vice d) is input in the case of the *Table* excitation.) The parameters d (or w) are expressed as the number of points, out of base total points, that define the diameter (or width) of the function

To transition from a diameter in terms of a number of points to an actual metric value, two equations were needed. The equations are

$$\Delta x = \frac{1}{2\rho_{\max}} \tag{4.1}$$

and

$$d = k\,\Delta x \tag{4.2}$$

$$= \frac{k}{2\rho_{\max}}, \tag{4.3}$$

where Δx is the length of a segment, $\rho_{\max}$ is the maximum spatial radius, k is the number of segments, and d is the diameter (or width w for the *Table* function). To determine the metric diameter, Eq. 4.1 was used to solve for Δx by setting $\rho_{\max}$ to 200 m^{-1}. This value resulted in a Δx of 2.5×10^{-3} m or 2.5 mm.

If the Gaussian excitation is selected, the user enters the value of the standard deviation σ, upon request. In a Bessel excitation selection the user enters the scaling factor a when requested. Table 4.2 gives a summary of the defining parameters used in AC_PROP.

Parameter	Definition (units)
*base**	Size of square base array
*slices**	Number of time slices
*Step**	Number of leading zero–rows
*c**	Acoustic velocity in media (m/s)
*z**	Distance, source–to–receiver (m)
*time_max**	Maximum time of propagation (s)
*rho_max**	Maximum spatial frequency (m^{-1})
d	Diameter of excitation function (sample points)
w	Width of *Table* excitation function (sample points)
σ	Gaussian standard deviation
a	Bessel scaling factor
* passed from AC_FIL.M	

Table 4.2: Summary of the defining parameters used in AC_PROP.

4.2 Program Verification

In verifying the program output, two excitation functions were used, the *Table* and the *Circle* functions. The outputs generated by the program from these excitations were compared to the results found with the previous FORTRAN implementations [4, 3, 7] for validation. After a general explanation about the generation, formatting, and titling of the outputs, the two excitation functions are presented. The *Table*, the first verification function, is then discussed and a table of defining parameters is given. The second verification function, the *Circle*, follows with a similar discussion and table of defining parameters.

4.2.1 Format of Results

The graphical outputs for the two excitation functions used for verification, the *Table* and the *Circle*, were generated, formatted, and titled in the same manner. The outputs are for a source–to–receiver distance of $z = 10$ cm. Each output was for a given time slice and consisted of a 128x128 array of values. There were 64 time slices including 3 leading all–zero arrays which simulate the step function at the arrival of the wave ($t = z/c$). The MATLAB programs contain optional commands to plot the output data using MATLAB's plotting routines or to store the data in ASCII format for importation into the AXUM plotting program. (The AXUM program is more flexible than the MATLAB plotting routines.)

Additionally, the 64 data arrays were combined with the Spyglass Data Utility into a 128x128x64 array. Spyglass Transform and Dicer could be used to interactively pick the data slice of interest and to plot the results.

NAME	VALUE	DEFINITION
base	128	Size of square base array
slices	64	Number of time slices
Step	3	Number of leading zero–rows
c	1500	Acoustic velocity in media (m/s)
z	0.1	Distance, source–to–receiver (m)
time_max	3.75×10^{-4}	Maximum time of propagation (s)
rho_max	200	Spatial radius of the filters (length^{-1})
w	23, 31	Width of *Table* (samples)

Table 4.3: Values of defining parameters for the *Table* input function used for program verification.

4.2.2 *Table* Impulse Excitation

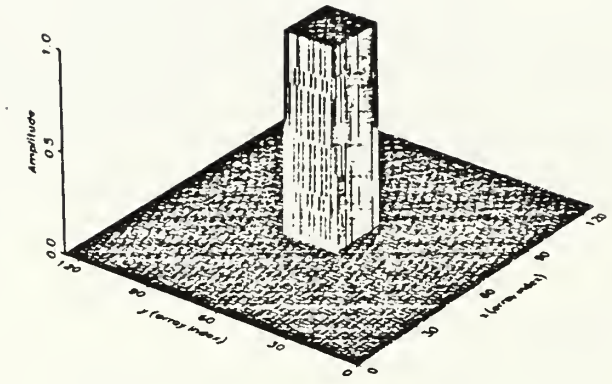
The first excitation function to be run by the program was the *Table* function (for a square piston source). There were several reasons to use the *Table* as the first input; the *Table* function is an easy function to implement and the results could be readily compared to results found in the literature [3, 5, 6, 7]. Table 4.3 provides list of the defining parameters used.

The values of *base*, *slices*, z, *time_max*, *rho_max*, and w chosen in Table 4.3 parallel those found in the literature used for validation. The acoustic velocity c of 1500 m/s is the velocity in water. To simulate the step function, a number *Step* of leading zero–rows was incorporated and arbitrarily set to three. The results were comparable to those in the literature and are presented (with the input functions) in Figs. 4.1 and 4.2.

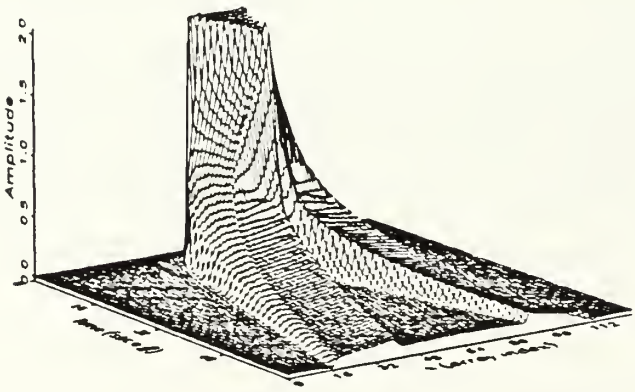
Here the widths $w = 23$ and $w = 31$ samples translate into metric widths of $w = 5.5$ cm and $w = 7.5$ cm, respectively. In the $w = 23$ case, $k = 22$ was the input in Eq. 4.3 and, in the $w = 31$ case, the input was $k = 30$ samples. Note that the x and y axes of the input functions range from 1 to 128, delineating a 128x128 base array.

The outputs present the magnitude of the acoustic potential at an observation point 10 cm from the source, $p(x, 0, 10, t)$. A diffraction duration from the initial impulse ($t = 0$) to $t =$*time_max* (150 μs) is represented as a function of radial distance; i.e., $p(x, 0, 10, 0)$ to $p(x, 0, 10, 150)$ is represented. This gives the 3–D view of the general diffraction through time. The output images show several interesting features.

The development of "tails," explained in terms of edge waves [3, 18], can be seen in the 3–D images. Also of note are the overshoots, having a maximum magnitude of 2.11 (both cases), and the undershoots (difficult to see in these two cases); these are due to the additive nature of the interference patterns of the waves originating on the edges of the discontinuous source.



(a)



(b)

Figure 4.1: *Table* spatial input and time–space output for $w = 23$ samples ($w = 5.5$ cm) at $z = 10$ cm.

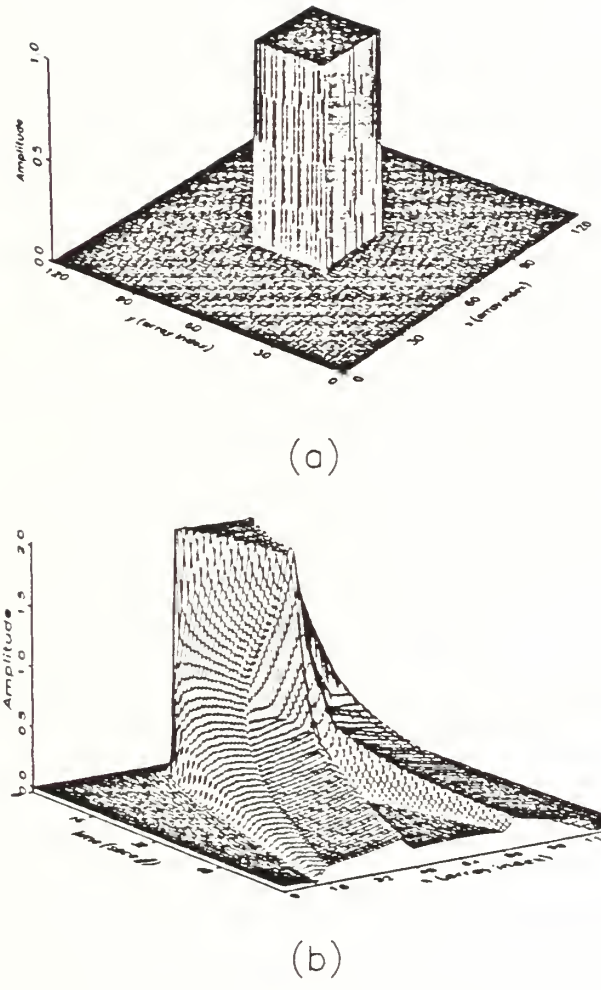


Figure 4.2: *Table* spatial input and time–space output for $w = 31$ samples ($w = 7.5$ cm) at $z = 10$ cm.

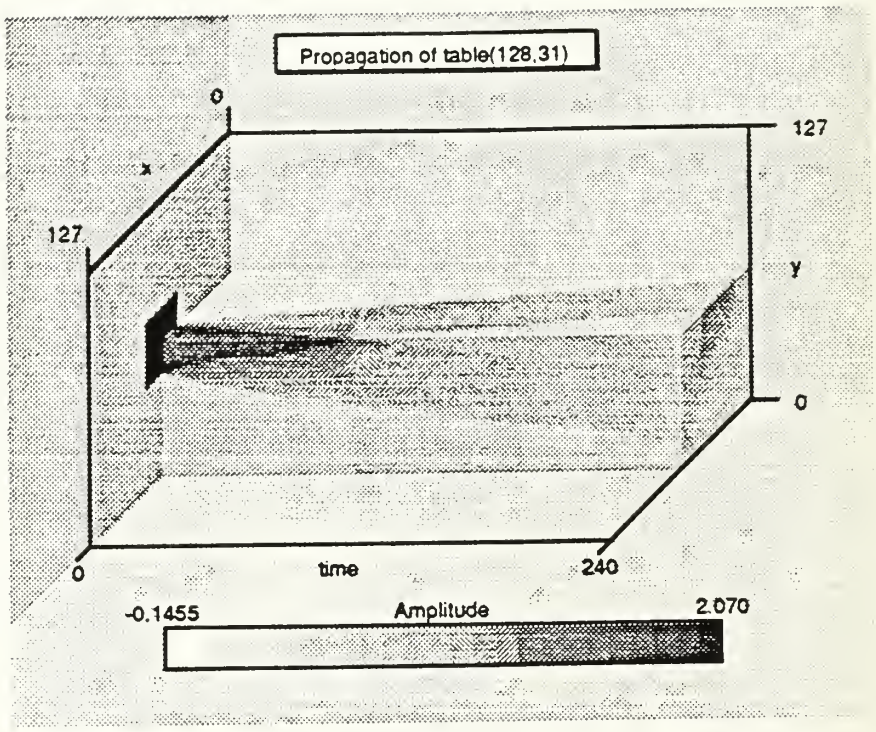


Figure 4.3: *Table* output for $d = 31$ samples. Dicer representation.

3D outputs for *Table* excitation

The Spyglass Dicer program allows selected three–dimensional representations of the propagation output. Figure 4.3 shows a rectangular cube set into one–fourth of the data volume. The x and y axes represent the spatial variables and range from 0 to 127 samples. The *time* axis represents the 64 time slices of data that were computed. To elongate the time axis, the Dicer program was used to increase the number of slices by a factor of 4x (to a total of 256 slices); linear interpolation is used to compute the values of the expanded slices lying between the original slices. Due to the nature of the interpolation used, only the expanded slices ranging from 0 to 252 are shown on the time axis. The three visible surfaces of the rectangular cube show the calculated data in three planes. The three planes are located at $x = 64$, $y = 65$, and $time = 252$. An additional plane is located at $time = 16$ to show the shape and size of the source (since the theory predicts that the field at $time = 16$ expanded samples [when $t = z/c$] duplicates the source excitation).

An alternative Dicer representation of the field is shown in Figure 4.4. Here, a horizontal slice of the data is placed at $y = 65$ and five vertical slices of the data are located at $x =$

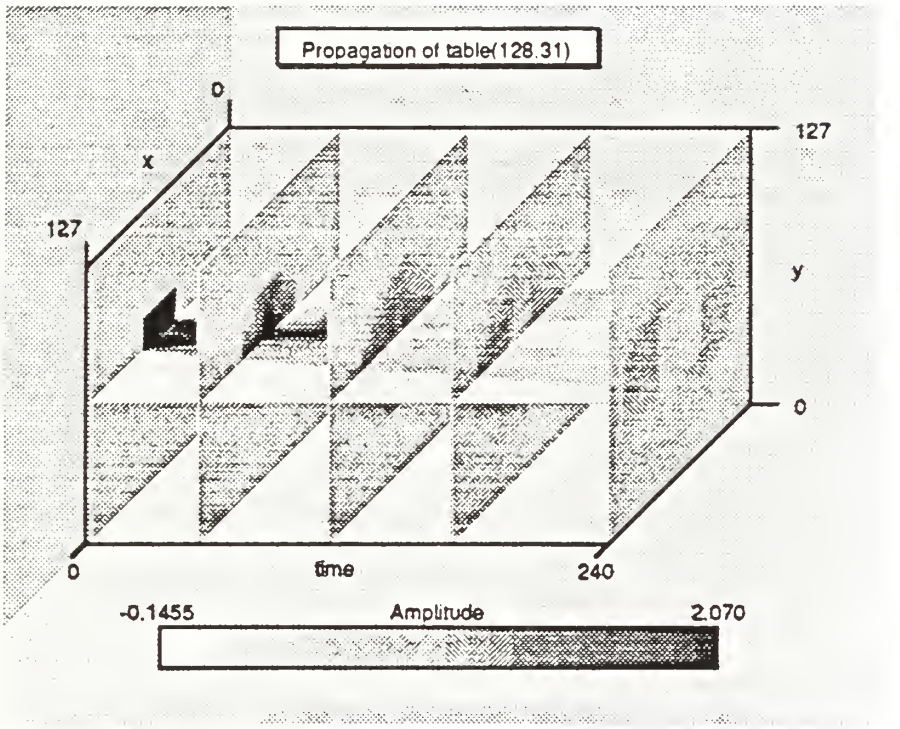


Figure 4.4: *Table* output for $d = 31$ samples. Alternative Dicer representation.

16, 64, 132, 200, and 252. (The values were arbitrarily chosen, but were restricted to be a multiple of four to avoid showing interpolated values of data.)

Appendix E contains Dicer representations for the other *Table* and *Circle* excitations that were studied.

4.2.3 *Circle* Impulse Excitation

The defining parameters for the *Circle* excitation are the same as those introduced in Table 4.3 with the exception that only the $d = 31$ case is presented. A diameter $d = 31$ samples translates into a metric diameter of $d = 7.5$ cm. Again the results are comparable to those found in the literature [5, 7]. Figure 4.5 gives the input function and the ensuing outputs. As in the *Table* case, the base array is a 128x128 array with the propagation pattern formed by successive $p(x, 0, 10, t)$ time slices. The results in Fig. 4.5 for the *Circle* excitation are much the same as those for the *Table* in Fig. 4.2. The "tails," however, shown in the 3–D image of Fig. 4.5 are rounded instead of cornered as in the *Table* output. Though the *Table* output holds its magnitude for a longer time, the maximum for the *Circle* output is greater at 2.21. (This value was read from the array; it is difficult to obtain quantitative information from the 3–d plots.) Also the drop off from the maximum is steeper for the *Circle*. The greater maximum and steeper drop off are due to the equal distance of all edge points from the center. This same geometric influence also accounts for the *Circle* holding the input value for a shorter duration. Just as the interference patterns added to a maximum greater than the input, the interference patterns also combine to give a more negative minimum. The negative undershoot was present for the *Table*; however, it has a magnitude five times greater for the *Circle*.

4.3 Other Input Excitations

Having checked the performance of the technique with piston sources, other sources with nonuniform spatial excitations were investigated The first is a circularly truncated Gaussian distribution function. Following the Gaussian, a circularly truncated Bessel profiled function is examined. These two excitation function outputs are generated and formatted the same.

4.3.1 Gaussian Distributed Excitation

Though the Gaussian has been investigated before [3]– [7], it has not been studied as a 128x128 array. The defining parameters for the case studied are listed in Table 4.4. Figure 4.6 shows the input and resulting output.

The Gaussian excitation function has been normalized by the maximum value of the computed Gaussian (see the Gaussian source code titled CRCGAUS.M in Appendix D). This normalization is shown in the input image of Fig. 4.6 by the maximum amplitude of one. Displayed as a *base* x *base* array, this input image has a standard deviation of $\sigma = 5$ and a 1/e point of 10.17 samples from the center (metric equivalent of $r = 2.54$ cm). The diffraction of this input is shown in Fig. 4.6.

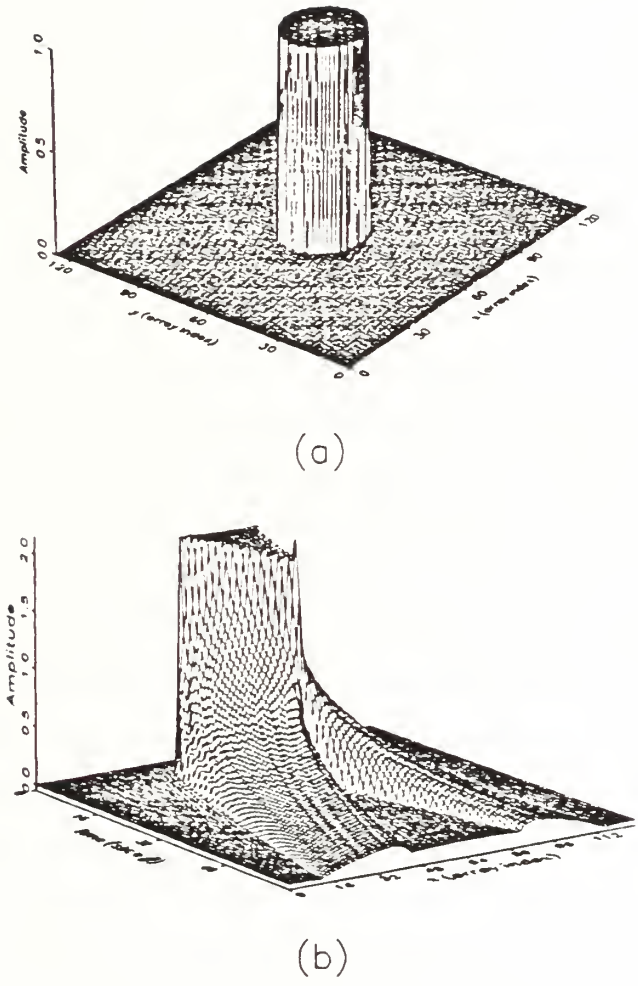
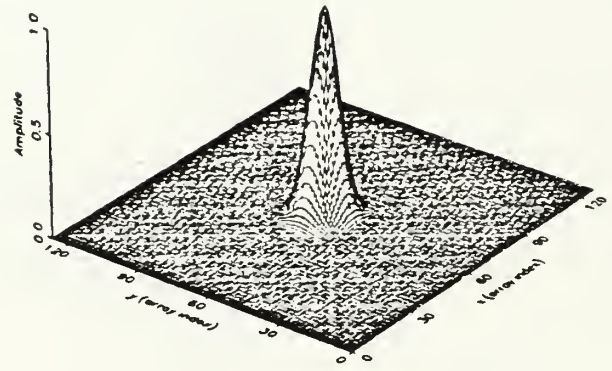
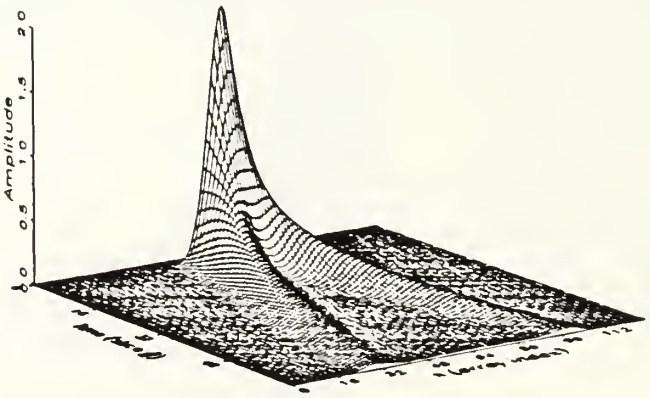


Figure 4.5: *Circle* input excitation and output for $d = 31$ samples.



(a)



(b)

Figure 4.6: Gaussian distributed input and output for $\sigma = 5$.

NAME	VALUE	SUMMARY
base	128	Size of square base array
slices	64	Number of time slices
Step	3	Number of leading zero–rows
c	1500	Acoustic velocity in media (m/s)
z	0.1	Distance, source–to–receiver (m)
time_max	375×10^{-6}	Maximum time of propagation (s)
rho_max	200	Spatial radius of filters (length^{-1})
d	51	Diameter of excitation function (samples)
σ	5	Gaussian standard deviation

Table 4.4: Defining parameters for Gaussian excitation case.

NAME	VALUE	SUMMARY
base	128	Size of square base array
slices	64	Number of time slices
Step	3	Number of leading zero–rows
c	1500	Acoustic velocity in media (m/s)
z	0.1	Distance, source–to–receiver (m)
time_max	375×10^{-6}	Maximum time of propagation (s)
rho_max	200	Spatial radius of filters (length^{-1})
d	51	Diameter of excitation function (samples)
a	0.25	Bessel scaling factor

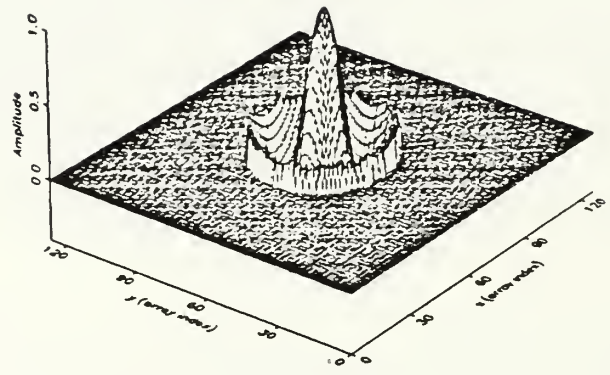
Table 4.5: Defining parameters for Bessel excitation case.

The 3–D image shows a diffraction pattern that is well established by time $t =$ *time_max*, forming two spreading "tails." The "tails," as well as the rest of the Fig. 4.6. diffraction pattern, are smoothly rounded. This rounding is the result of the continuity of the Gaussian distribution. A discontinuity, as in the previous two excitation shapes, results in a characteristic over and undershoot of the maximum and minimum inputs. Again, the results for this Gaussian excitation conformed to those found in the literature [3, 4, 5, 6, 7]. The Bessel excitation was then run and the results compared to those of the Gaussian.

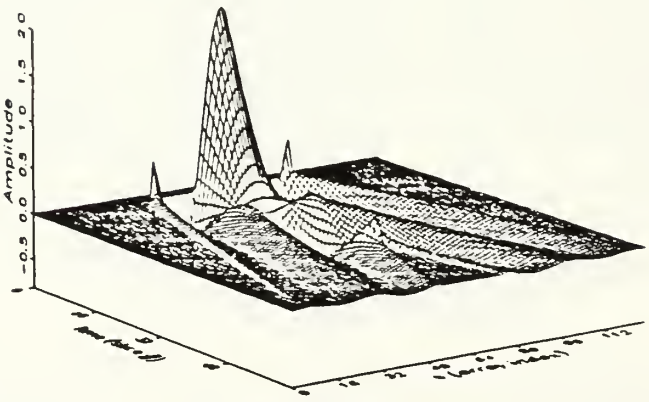
4.3.2 Bessel Excitation

Results for the Bessel excitation produced by CRCBES.M were generated, as previously discussed, for the set of defining parameters listed in Table 4.5.

The resulting input and output are shown in Fig. 4.7. Since the output is formed by



(a)



(b)

Figure 4.7: Bessel-profile input and output for $a = 0.25$.

taking successive $p(x, 0, 10, t)$ vectors, three peaks appear in the 3–D image. These three peaks correspond to the center peak and the points on the two crests directly adjacent to the center. As a result of having three peaks, three "tails" are present. The "tails," however, are smooth because a Bessel function is a continuous function. Also worth noting in the 3–D image is the simulated step function, more visible here due to the oscillations of a Bessel.

Comparing the Gaussian and Bessel outputs, it is seen that the Gaussian's magnitude retention is slightly longer than that of the Bessel. This is due to the more gradual decrease in magnitude vice the steeper decrease required of the Bessel so that it can become negative. Still no hard conclusions can be drawn without further analysis.

Chapter 5

Summary

This report presented a MATLAB implementation of a Fourier approach to acoustic wave propagation. A mathematical development using linear systems that found the acoustic potential from an arbitrary spatial and temporal source was reviewed. In the mathematical development, it was shown that the Green's function solving the appropriate wave equation and satisfying the boundary conditions is the total impulse response of the system. Through double and triple convolutions, the acoustic potential could be found for any source separable in time and space. Use of the 2–D spatial Fourier transform, however, translated the convolution to multiplication in the spatial frequency domain. This made the MATLAB implementation easier.

After an overview of MATLAB and the graphics program AXUM, a functional description of the program was furnished. The program modules AC_FIL and AC_PROP both made use of symmetry. AC_FIL generated the time–varying filters, the most time consuming process, while AC_PROP accomplished the remaining computations making use of MATLAB's "fft2" function. Details of both modules as well as the source code have been included in the Appendix A.

Several examples were delineated in the body of this thesis. First, the *Table* and the *Circle* were presented as the program verification excitations; the results conformed to those found in the literature. Then two newer excitation functions, the Gaussian and the Bessel, were presented.

The underlying result was an accurate and efficient computer implementation of the linear systems approach to ultrasonic wave propagation. The efficiency was derived from the modularization of the program so that consecutive runs could be made without recomputing the most time consuming portion, the filters. Also, the use of MATLAB's "fft2" function bypassed tedious and time–inefficient convolution integrals. Finally, both modules made use of symmetry by computing only one quadrant of data which was then manipulated into the remaining quadrants. An advantage to using MATLAB was the ease of expansion that could be accomplished with the program.

The work of this report concentrated on a source with rigidly baffled boundary conditions and a lossless media. Cases that include free space and resiliently baffled boundary

conditions as well as lossy media, linear and quadratic lossy media, could be incorporated. A few facts worth noting here are that the free space and resilient baffle boundary conditions can be expressed in terms of the rigid baffle case [4] and that the lossy media and lossless media transfer functions are interdependent [6]. Furthermore, new excitation functions, such as a phased array or a focused source [2], could also be incorporated. Improvements are needed in the area of analysis such as the Gaussian versus Bessel propagation comparison and extending the technique to sources that are not time and space separable such as new non–diffracting waves [19].

5.1 Acknowledgements

This work was supported by the Direct–Funded Research program at the Naval Postgraduate School. The author would like to acknowledge the contributions of Dr. Daniel Guyomar, who developed the mathematical theory of the method while serving as a National Research Council Postdoctoral Associate at the Naval Postgraduate School. The programming contributions and insights offered by LT Tim Merrill, USN; LCOL John Upton, USMC; and LT Bill Reid, USN, are gratefully recognized. In addition, our lab engineer, Mr. Ray van de Veire, eased the programming and plotting of the outputs.

Appendix A

Source Code for AC_FIL.M

The following is the source code used to generate the time–varying filters as discussed in Chapters II and III. AC_FIL.M was written in block format with each block headed by a descriptive comment to explain the block's function. The code includes many optional instructions indicated by the leading %* symbol. Deleting this symbol will enact that line of code on succeeding program runs which, in turn, varies the output. The outputs necessary to a successful run of AC_PROP.M are the files, AC*base*x($m + Step$).MAT (where m is an index number from 1 to 61). and the file, AC_FIL.MAT. For example, the file, AC128x08.MAT, contains the data for the propagation spatial filter in a 128x128 array for the fifth time slice (since Step = 3). The file, AC_FIL.MAT, contains the parameters needed in AC_PROP.M.

Code is provided for both the DOS version and the Sun workstation version of MATLAB4. There are two primary differences in the versions. First, the paths to disk storage are different, to reflect the path setup on the two host machines. Secondly, the commands to obtain a hard–copy version of MATLAB4's graphics differed. In the DOS version, the user prints copies from the menu associated with the graphics window or with a **print** command. In the SUN version, a hard copy can be printed only by a **print** command. Each method is included in the file text; the user selects the appropriate command by removing the "the desired lines.

AC_FIL.M SOURCE CODE

```
%%%%          **** AC_FIL.M ****
%%
%% This program generates an Acoustic Propagation Transfer
%% Function, a time varying spatial filter, for use in
%% AC_PROP.M to simulate acoustic wave diffraction.
%%       William H. Reid          December 1992
%%       Modified for MATLAB4 and Sun. 9/93 JPP

clear;                          % Clear MATLAB
```

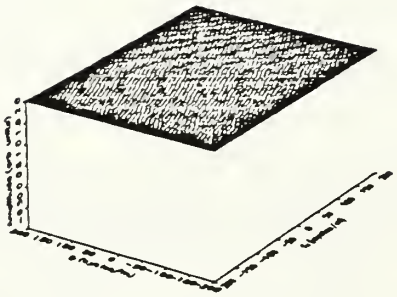
```
tm1=clock; ;                    % Start timer clock

base = 128;                     % Size of square base array.
N0 = (base/2)+1;                % Defines center of square base array.
slices = 64;                    % Number of time slices.
Step = 3;                       % Number of leading zero time slices,
                                % simulates the step function.
c = 1500;                       % Velocity of the acoustic wave, (m/s).
z = 0.1;                        % Distance to the observation plane, (m).
time_max = 3.75e-4;             % Maximum time of propagation,
                                % time at the final time slice, (sec).
rho_max = 200;                  % Spatial radius of the filter,
                                % [sqrt(rhox^2 + rhoy^2]. (per length)

%% Initialize arrays to reduce processing time.
  shft_filter = zeros(base);  temp = zeros(N0);
  arg = zeros(N0);  rhom = zeros(N0,1);
  rho = zeros(N0);  time = zeros(slices-Step,1);

%% Generate "slices-Step" time slices between z/c and time_max.
  time = linspace(z/c,time_max,slices-Step);

%% Choose directory to store data.
cd i:/ac_prop/filters                          % PC version
%* cd /home2/powers/M_files/ac_prop/filters    % SUN version
%% Save variables necessary for passing to AC_PROP.M in AC_FIL.MAT.
 save ac_fil  base N0 slices Step c z time_max rho_max;

%% Generate N0-1 values of "rhom" from 0 to rho_max.
  rhom = linspace(0,rho_max,N0-1);

%% Add additional increment to rhom to compensate for off-center
%% orientation of the final  base x base matrix.
  rhom = [rhom (rhom(N0-1)+rhom(2))];

%% Create two N0 x N0 arrays of rho values for function evaluation.
  [rhox,rhoy] = meshgrid(rhom,rhom);

%% Calculate "rho", an N0 x N0 matrix of radial distances for use in
%% the argument to the Bessel function within the loop.
  rho= sqrt(rhox.^2 + rhoy.^2);
```

```
%%%%%%%%%%% STARTLOOP %%%%%%%%%%
%% Generate "slices-Step" filter arrays, the filter at each time slice.
  for m = 1:(slices-Step)
    fprintf('%3.0f',m);      % Display m value for user progress report.

%% Create an N0 x N0 array of argument values for the Bessel function.
    arg = rho * sqrt(c^2*time(m)^2-z^2 );

%% Evaluate the zero order Bessel for each argument value;
%% creates an N0 x N0 array called "temp".
    temp = 2*besselj(0,arg);

%% Create shft_filter matrix containing the spatial filter.
%%  (Done by flipping "temp" into all quadrants.)
shft_filter(N0:base,N0:base) = temp(1:N0-1,1:N0-1);
shft_filter(N0:base,1:N0-1) = fliplr(temp(1:N0-1,1:N0-1));
shft_filter(1:N0-1,1:N0-1) = temp(N0:-1:2,N0:-1:2);
shft_filter(1:N0-1,N0:base) = flipud(temp(2:N0,2:N0));

%% Save shft_filter in a file named "a(base)x(m+Step).mat",
    cd i:/ac_prop/filters                          % PC version
%*  cd /home2/powers/M_files/ac_prop/filters       % SUN version

if (m+Step) < 10,
   %% MATLAB format
   eval(['save a',int2str(base),'x0',int2str(m+Step),' shft_filter m' ] );
   %% ASCII format
   %* eval(['save a',int2str(base),'x0',int2str(m+Step),'.dat shft_filter...
   %*       /ascii']);
     else
   %% MATLAB format
   eval(['save a',int2str(base),'x',int2str(m+Step),' shft_filter m' ] );
   %% ASCII format
   %* eval(['save a',int2str(base),'x',int2str(m+Step),'.dat shft_filter...
   %*      /ascii']);
     end

 end
%%%%%%END LOOP%%%%%%%%%

toc;                                  % Stop timer clock
time_elapsed = etime(tm2,tm1)/60      % Compute & display execution time
```

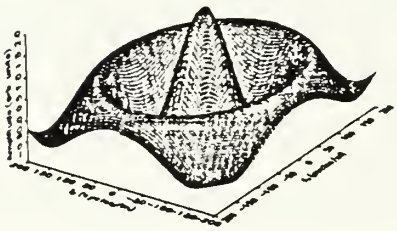
Appendix B

Examples of the Time–varying Propagation Bessel Filters

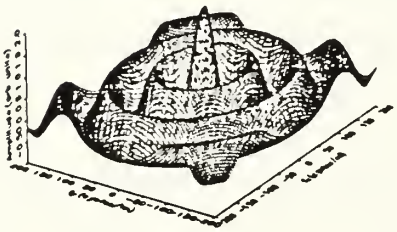
This appendix contains examples of the time–varying filters (Eq. 3.1 on page 12) produced by AC_FIL for the parameters described in the text. Only a few of the 61 total slices are shown. The first slice, at $t = z/c$, is a plane with amplitude two; this is because the input argument to the Bessel function is zero giving an output value of one. This uniform plane produces an output for time slice one that is a scaled replica of the input. The remainder of the filters illustrate the time variance and how the filters collapse inward with time. Each filter is a 128x128 array representation in the spatial frequency domain.



(a)



(b)



(c)

Figure B.1: Propagation filter at time slices 1, 2, and 5.

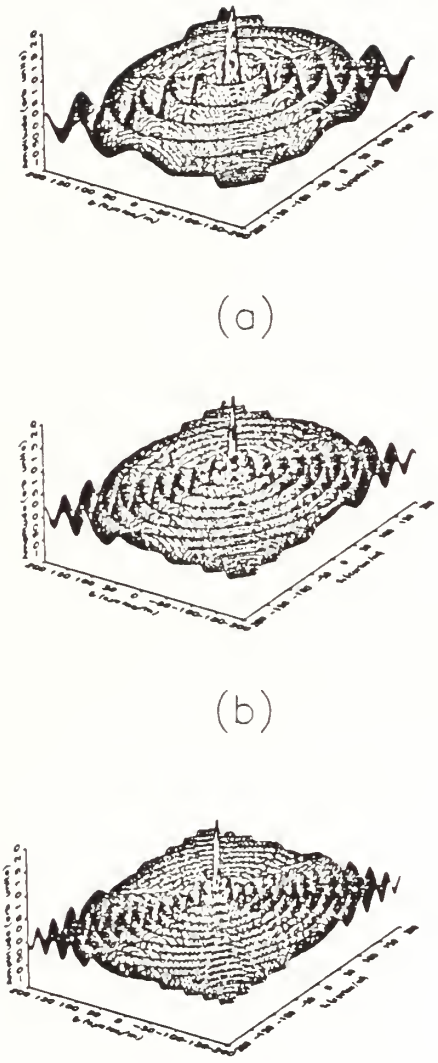


Figure B.2: Propagation filter at time slices 10, 20, and 30.

Appendix C

Source Code for AC_PROP.M

The following is the source code for AC_PROP.M that was used to produce various outputs, including those discussed in Chapter IV. AC_PROP was written in blocks with each block headed with a descriptive comment to explain the block's function. This also makes it easy to follow the computation from inputs to output. Inputs to AC_PROP are imported from the file AC_FIL.MAT and the files A*basexm* + *Step*.MAT (*m* is an index number ranging from 01 to 61). The program solicits user input to determine the input excitation. AC_PROP then computes the output.

The format of the output can be changed by the user. Before running the program, the user can remove the optional comment markers indicated with %% to produce and/or view the output in the desired format. This allows the data to be saved and exported in ASCII format for use with other graphics programs (such as AXUM, see Chapter II).

AC_PROP.M SOURCE CODE

```
%%%%%                **** AC_PROP.M ****
%%
%% This script file performs transient-wave acoustic propagation
%% simulations. It uses the time-varying spatial filters called
%% "shft_filter" found in the filters directory under "a(base)x(m).mat"
%% to compute the acoustic propagation fields. The "shft_filter"
%% data are generated by AC_FIL.M.
%%
%%       William H. Reid, December 1992
%%       Modified for MATLAB4 and Sun. 9/93 JPP

clear; clc;        % Clear MATLAB
tic;               % Start timer
format compact     % Set compact format for screen display.
```

```
%% Load the parameters generated by last run of AC_FIL.M.
%% PC and SUN directories have different names
   cd h:\ac_prop\filters                        % PC version
%* cd /home2/powers/M_files/ac_prop/filters  % SUN version
load AC_FIL

%% Display the size of the square base array.
disp('base is the width of the array.'); disp('   '); base

%% Generate the INPUT function from user interface.
input_func = menu('Choose the shape of input function.','Circle', ...
       'Square','Truncated Gaussian','Truncated Bessel');
if isempty(input_func); input_func = 3; end  % Default to Gaussian input.
if input_func == 1,      % Cicle input
       name='c';
       disp('You have chosen a truncated circle input.')
       d = input('Please enter an ODD diameter, [51], d =  ');
        if isempty(d); d = 51; end  % Default diameter: 51 samples
       shft_input = crcle(d,base);    % Create Circle input
       % Name a file to hold the input function.
       InFile_Name = [name,'i',int2str(base),'x',int2str(d)];
       % Name a file to  hold the output.
       OutFile_Name = [name,'o',int2str(base),'x',int2str(d)];
 elseif input_func == 2,   % Square input
       name='s';
       disp('You have chosen a truncated square input.')
       w = input('Please enter an ODD width, [11], w =  ');
        if isempty(w); w = 11; end  % Default width: 11 samples
       shft_input = table(w,base);    % Create input
        d = w;
       % Name a file to hold the input function.
       InFile_Name = [name,'i',int2str(base),'x',int2str(d)];
       % Name a file to  hold the output.
       OutFile_Name = [name,'o',int2str(base),'x',int2str(d)];
 elseif input_func == 3,    % Gaussian input
       name='g';
       disp('You have chosen a truncated Gaussian input.')
       sigma = input('Please enter the standard deviation, [10], sigma =  ');
        if isempty(sigma); sigma = 10;  end  % Default sigma: 10 samples
       d = input('Please enter an ODD diameter, [51], d =  ');
        if isempty(d); d = 51; end  % Default diameter: 51 samples
       shft_input = crcgaus(sigma,d,base);  % Calculate input
```

```
        % Name a file to hold the input function.
        InFile_Name = [name,'i',int2str(base),'x',int2str(d)];
        % Name a file to  hold the output.
        OutFile_Name = [name,'o',int2str(base),'x',int2str(d)];
elseif input_func == 4,      % Bessel input
        name='b'
        disp('You have chosen a truncated Bessel input.')
        a = input('Please enter a width scaling factor,[0.3125], a =  ');
            if isempty(a); a = 0.3125; end % Default: 0.3125
        d = input('Please enter the ODD diameter, [51], d =  ');
            if isempty(d); d = 51; end  % Default: 51 samples
        disp('Please wait. This calculation takes a while.....')
        shft_input = crcbess(a,d,base);  % Calculate input
        q = a * 1e4;
        % Name a file to hold the input function.
        InFile_Name = [name,'i',int2str(base),'x',int2str(q)];
        % Name a file to  hold the output.
        OutFile_Name = [name,'o',int2str(base),'x',int2str(q)];
 lse
        disp('   ');
        disp('Incorrect Excitation Function Selection!');
        error('Restart AC_PROP...to try again.');
end

%% Save the input function placed in "InFile_Name" in ascii format
%% for use with other graphics software.
 eval(['save ',InFile_Name,'.dat shft_input /ascii']);
 clear InFile_Name;       % Remove "InFile_Name" from memory.

%% Compute the sample spacing in x and y (delta_x) and the spacing in
%%    time (delta_t).
 delta_x = (base-2)/(2*rho_max); delta_t = time_max/(slices-Step);
%% Create X and T vectors (1 x base).
 X = linspace(-(NO-1)*delta_x,((base-NO)-1)*delta_x,base);
 T = linspace(-3*delta_t,time_max,slices);

%% Display the input function.
%* mesh(X,X,abs(shft_input));title('|SHFT_INPUT|');
%* axis( [ X(1) X(base) X(1) X(base) ...
%*      min( [ 0 min(min(abs(shft_input))) ] ) max(max(abs(shft_input))) ])
%* xlabel('x (cm)'); ylabel('y (cm)'); zlabel('amplitude'); view(52.5,30);
```

```
%% SUN: Save input figure as eps file.
%*    cd /home2/powers/M_files/ac_prop/data
%%  PC: save figure from figure window.
%*    cd h:/ac_prop/

%% Save input plot as EPSF file with bitmap preview image.
%*    disp(' '); disp('Saving input plot as EPS file.....'); disp(' ');
%*    eval(['print -deps -epsi ',name,'_in_b',int2str(base), ...
%*              '_d',int2str(d)]);
%*    disp(' '); disp('Saving input plot as EPS file.....'); disp(' ');
%*    eval(['print -deps -epsi ',name,'_in_b',int2str(base), ...
%*              '_d',int2str(d)]);

%% Shift "shft_input" from centered geometry to corner geometry and take
%%    2-D FFT to produce F_INPUT.
 F_input = fft2( fftshift(shft_input) );
 clear shft_input;                          % Free RAM.

%% Shift F_input in preparation of multiplication with PROP_m.
 shft_F_input = fftshift(F_input);
 clear F_input;   % Free RAM.

%% Element by element arrray multiplication of the transfer function
%% filter in "PROP_m" and  "Fshft_input."
 disp('Performing array multiplication....');


%* cd /home2/powers/M_files/ac_prop/data % SUN version
   cd h:/ac_prop/data                   % PC version

%% Save "Step" arrays full of zeros for first array values.
%%%%%%%%%%%%% START FIRST LOOP %%%%%%%%%%%%%%%
for m=1:Step
   shft_output = zeros(base);
%% Save the (base)x(base) array "shft_output" in an ASCII file whose
%% name is  "(input_func)_OUT_(m).dat".
   eval(['save ',int2str(input_func),'_OUT_0',int2str(m), ...
'.dat shft_output /ascii']);
   clear shft_output;       % Get ready for next pass.
end
%%%%%%%%%%%%%  END FIRST LOOP  %%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%% START SECOND LOOP %%%%%%%%%%%%%%%
for m = 1:(slices-Step)
    fprintf('%2.0f,',m)      % Display "m" value for progress report.

%% Give the variable "filename1" the name of the file containing
%% "shft_filter" and then load that file.
    if m < 10-Step,
      filename1 = ['a',int2str(base),'x0',int2str(m+Step)];
    else
      filename1 = ['a',int2str(base),'x',int2str(m+Step)];
    end
%* cd /home2/powers/M_files/ac_prop/filters % SUN version
   cd h:/ac_prop/filters                    % PC version
   eval(['load ',filename1] );

%% Multiply the filter by the shifted transform of the input.
   shft_F_output = (shft_filter .* (shft_F_input));
   clear shft_filter;                       % Free RAM

%% Shift "shft_F_output" from centerd geometry to corner geometry
%%    ("F_output") and take inverse transform to produce "output."
    output =  ifft2( fftshift(shft_F_output) ) ;
%% Shift "output" to center geometry ("shft_output").
    shft_output = fftshift(output);
%% "shft_output" is the centered geometry version of the diffracted
%%       wave at time slice "m".
    clear shft_F_output output              % Free RAM.

%% Place the transposed "N0" (center) row of "shft_output" in the
%% (m+Step)-th column of "output_plot."
%% This creates a (base) x (m+Step) array whose columns show a slice
%% of the diffracted wave.
    output_plot(1:base,m+Step) = (shft_output(N0,1:base))';

%% Save the (base)x(base) array "shft_output" in an ASCII file whose name is
%% "(input_func)_OUT_(m+Step).dat".
%% A Gaussian, for example, would be  G_OUT_12.dat on the 9-th loop
%% interation (Step = 3). This is optional for  graphics use by
%% other software.
%* cd /home2/powers/M_files/ac_prop/data % SUN version
   cd h:/ac_prop/data                    % PC version
    if m < 10-Step,
```

```
        eval(['save ',int2str(input_func),'_OUT_0',int2str(m+Step), ...
'.dat shft_output /ascii']);
     else
        eval(['save ',int2str(input_func),'_OUT_',int2str(m+Step), ...
'.dat shft_output /ascii']);
     end
   clear shft_output;       % Get ready for next pass.
 end
%%%%%%%%%%%%%%%% End loop %%%%%%%%%%%%%%%%

%% Save the contents of "output_plot" in a MATLAB file
%% and as an ascii file (optional).
%* cd /home2/powers/M_files/ac_prop/data % SUN version
   cd h:/ac_prop/data                    % PC version
 eval(['save ',OutFile_Name,' output_plot'])
 eval(['save ',OutFile_Name,'.dat output_plot /ascii']);

%% Display output view #1.
%%   figure; mesh(T,X,abs(output_plot)); title('|OUTPUT|');
%%   axis( [ T(1) T(slices)  X(1) X(base)  ...
%%        min( [ 0 min(min(abs(output_plot))) ] ) ...
%%        max(max(abs(output_plot))) ])
%%   xlabel('time (s)'); ylabel('y (cm)'); zlabel('abs(output)');
%%   view(52.5,30);

%%  Save output figure as eps file.
%* cd /home2/powers/M_files/ac_prop/data % SUN version
   cd h:/ac_prop/data                    % PC version
%*  disp(' '); disp('Saving output figure as EPS file.....'); disp(' ');
%*  eval(['print -deps -epsi ',name,'_out1_b',int2str(base), ...
%* '_d', int2str(d)]);

%% Create a different view of output
%*   figure; mesh(T,X,abs(output_plot)); title('|OUTPUT|');
%*   axis( [ T(1) T(slices)  X(1) X(base)  ...
%*        min( [ 0 min(min(abs(output_plot))) ] ) ...
%*        max(max(abs(output_plot))) ])
%*  xlabel('time (s)'); ylabel('y (cm)'); zlabel('abs(output)');

%% Save second output figure as eps file.
%* disp(' '); disp('Saving figure as EPSF file.....'); disp(' ');
%* eval(['print -deps -epsi ',name,'_out2_b',int2str(base),'_d', ...
```

```
%*      int2str(d)]);

elapsed_time = toc/60 % Stop timer; display elapsed time.
disp('minutes')
```

Appendix D

Source Code for Input excitations

The following source code is for the input excitation choices given to the user in AC_PROP. Each is written as a MATLAB function and can be used independently of AC_FIL or AC_PROP. The input excitations are the uniform square (*TABLE(w,base)*), the uniform circle (*CRCLE(d,base)*), the circularly truncated Gaussian (*CRCGAUS(sigma,d,base)*), and the circularly truncated Bessel (*CRCBES(a,d,base)*) where w is the width of the square, d is the diameter of the circle, σ is the standard deviation of the Gaussian distribution, a is a scaling factor, and *base* is the size of the base array.

TABLE.M SOURCE CODE

```
function Y = table(w,base)
%     TABLE.M: Y = table(w,base)
%Program for generating a uniform square excitation function.
%   December 1992     William H. Reid
%   Based on TABLE.M by JG Upton
%
%   w is the WIDTH of the table.  (ODD integer)
%   base is the WIDTH of the square base.  (EVEN integer)
%   Example:  z = table(33,64);

% Check that w is an odd integer.
      if rem(w,2) < 0.1;
      error('The width of the table must be an ODD integer.');
      else;
      end;

% Check that base is an even integer.
      if rem(base,2) ~= 0.0;
      error('The width of the square base must be an EVEN integer.');
```

```
      else;
      end;

N0 = (base/2)+1;                          % N0 is the base array's center.
w0 = ceil(w/2);                          % w0 is the mid--point of the table.

Y = zeros(base);                           % Initialize matrices to reduce
temp = zeros(N0-1);                       % processing time.

temp(1:w0,1:w0) = ones(w0);               % Set amplitude to one.

% Generate the entire base x base input function array.
Y(base0:base,base0:base) = temp;
Y(2:base0,base0:base) = rot90(temp);
Y(2:N0,2:N0) = rot90(temp,2);
Y(N0:base,2:N0) = rot90(temp,3);

% To test input distribution:  mesh(Y)
```

CRCLE.M SOURCE CODE

```
function Y = crcle(d,base)
%     CRCLE.M:  Y = crcle(d,base)
% Program for generating uniform circular excitation functions
%   December 1992     William H. Reid
%   Based on CRCLE.M by JG Upton
%
%    d is the DIAMETER of the circle.  (ODD integer)
%    base is the WIDTH of the square base. (EVEN integer)
%    Example: z = crcle(33,64);

% Check that d is an odd integer.
     if rem(d,2) < 0.1;
     error('The diameter of crcle function must be an ODD integer.');
     else;
     end;

% Check that base is an even integer.
     if rem(base,2) ~= 0.0;
     error('The width of the square base must be an EVEN integer.');
```

```
     else;
     end;

N0 = (base/2)+1;                        % N0 is the base array's center.
r = d/2;                                % r is the circle's radius.

% Initialize matrices to reduce processing time.
Y = zeros(base);
temp = zeros(N0-1);

% Set amplitude to one inside the circle's radius.
for m = 1:r+1;
    for n = 1:r+1;
        if sqrt((m-1)^2 + (n-1)^2) <= r;
        temp(m,n) = 1;
        end;
    end;
end;

% Generate the entire base x base input function.
Y(base0:base,base0:base) = temp;
Y(2:base0,base0:base) = flipud(temp);
Y(2:N0,2:N0) = rot90(temp,2);
Y(N0:base,2:N0) = fliplr(temp);

% To test input function distribution:  mesh(Y)
```

CRCGAUS.M SOURCE CODE

```
function Y = crcgaus(sigma,d,base)
%     CRCGAUS.M:  Y = crcgaus(sigma,d,base)
% Program for generating circular Gaussian excitation functions.
%   December 1992     William H. Reid
%   Based on CRCGAUS.M by JG Upton
%
%  sigma is the STANDARD DEVIATION of the Gaussian function.
%  d is the  DIAMETER of circle.  (ODD integer)
%  base is the WIDTH of the square base. (EVEN integer)
%  Example: z = crcgaus(12,33,64);

mu=0;                           %mu is the mean of the Gaussian function.
```

```
% Check that d is an odd integer.
      if rem(d,2) < 0.1;
      error('The diameter of circle function must be an ODD integer.');
      else;
      end;

% Check that base is an even integer.
      if rem(base,2) ~= 0.0;
      error('The width of the square base must be an EVEN integer.');
      else;
      end;

N0 = (base/2)+1;                % N0 is center of the array.
r = d/2;                      % r is the radius of the truncating circle.


% Initialize the matrices to reduce processing time.
Y = zeros(base);
temp = zeros(N0-1);

% Compute the amplitude for the Gaussian distributed circle.
for m = 1:(d+1)/2;
    for n = 1:(d+1)/2;
        x = sqrt((m-1)^2+(n-1)^2);
        if x <= r;
        temp(m,n) = (1/(sqrt(2*pi)*sigma))*exp(-((x-mu)^2)/...
                    (2*(sigma^2)));
        end;
    end;
end;

% Generate the entire base x base input array.
Y(base0:base,base0:base) = temp;
Y(2:base0,base0:base) = flipud(temp);
Y(2:N0,2:N0) = rot90(temp,2);
Y(N0:base,2:N0) = fliplr(temp);

Y = Y ./ (max(max(Y)));  % Normalize the Gaussian distribution to one.

% To test and view the input function:  mesh(Y)
```

CRCBESS.M SOURCE CODE

```
function Y = crcbess(a,d,base)
%     CRCBESS.M:  Y = crcbess(a,d,base)
% Program for generating circular Bessel excitation functions.
%   December 1992     William H. Reid
%   Based on CRCBESS.M by JG Upton
%
%    a is the WIDTH SCALING FACTOR.
%    d is the DIAMETER of the circle.  (ODD integer)
%    base is the WIDTH of the square base. (EVEN integer)
%    Example: z = crcbess(1,33,64);

% Check that d is an odd integer.
       if rem(d,2) < 0.1;
          error('The diameter of the circle must be an ODD integer');
       else;
       end;

% Check that base is an even integer.
       if rem(base,2) ~= 0.0;
          error('The width of the square base must be an EVEN integer');
       else;
       end;

N0 = (base/2)+1;                          % N0 is the center of the array.
r = d/2;                                  % r is the radius of the circle.

Y = zeros(base);                          % Initialize the arrays to reduce
temp = zeros(N0-1);                       %  processing time.

% Compute the Bessel distributed amplitude within the circle.
for m = 1:r+1;
    for n = 1:r+1;
        x = sqrt((m-1)^2 + (n-1)^2);
        if x <= r;
        temp(m,n)=besseln(0,a*x);
        end;
     end;
 end;

% Generate the entire base x base input array.
```

```
Y(N0:base,N0:base) = temp;
Y(2:N0,N0:base) = flipud(temp);
Y(2:N0,2:N0) = rot90(temp,2);
Y(N0:base,2:N0) = fliplr(temp);

% To test and view the input function:  mesh(Y)
```

Appendix E

Examples of Dicer Representations of Output Data

This appendix contains more representations of the data from the Spyglass Dicer program. The representations are discussed in the text on page 4.3.

Figures E.1 and E.2 show representations of data from a *Table* excitation that is 23 samples wide. Figures E.3 and E.4 show representations of data from a *Circle* excitation that is 23 samples wide. Figures E.5 and E.6 show representations of data from a *Circle* excitation that is 31 samples wide.

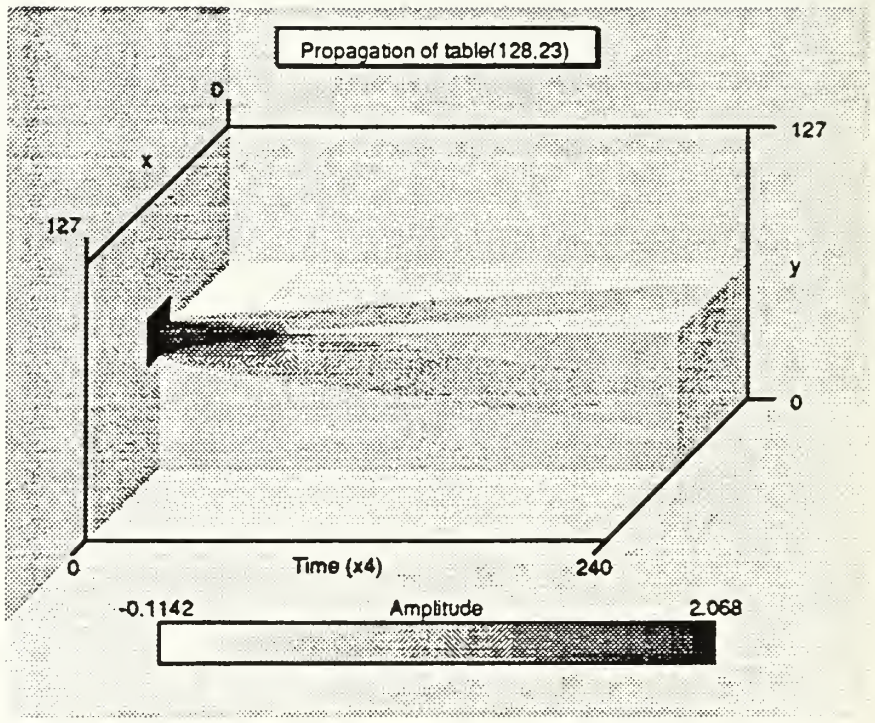


Figure E.1: *Table* output for $d = 23$ samples. Dicer representation.

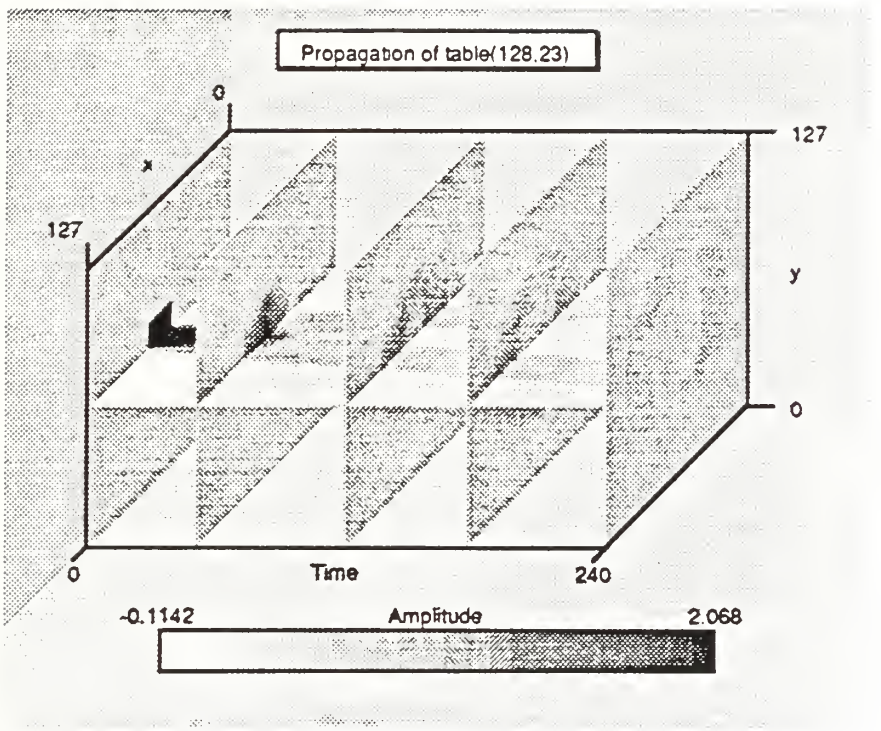


Figure E.2: *Table* output for $d = 23$ samples. Alternative Dicer representation.

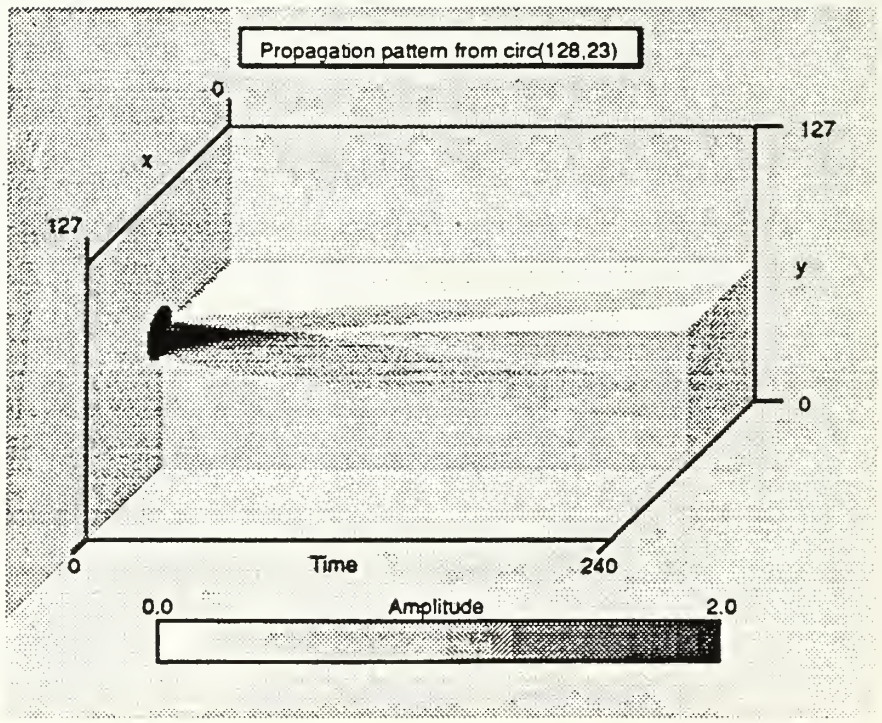


Figure E.3: *Circle* output for $d = 23$ samples. Dicer representation.

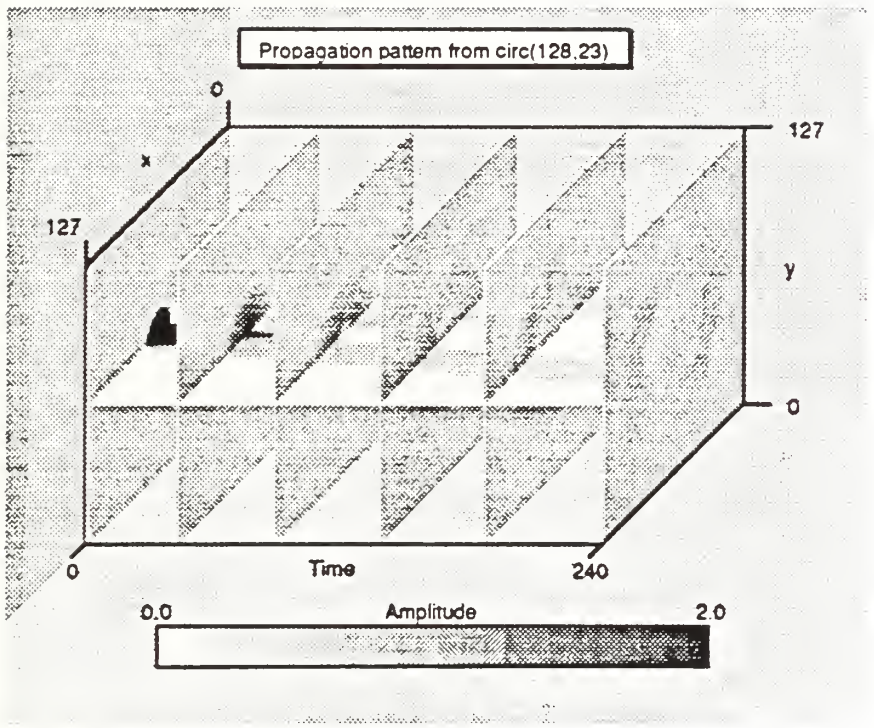


Figure E.4: *Circle* output for $d = 23$ samples. Alternative Dicer representation.

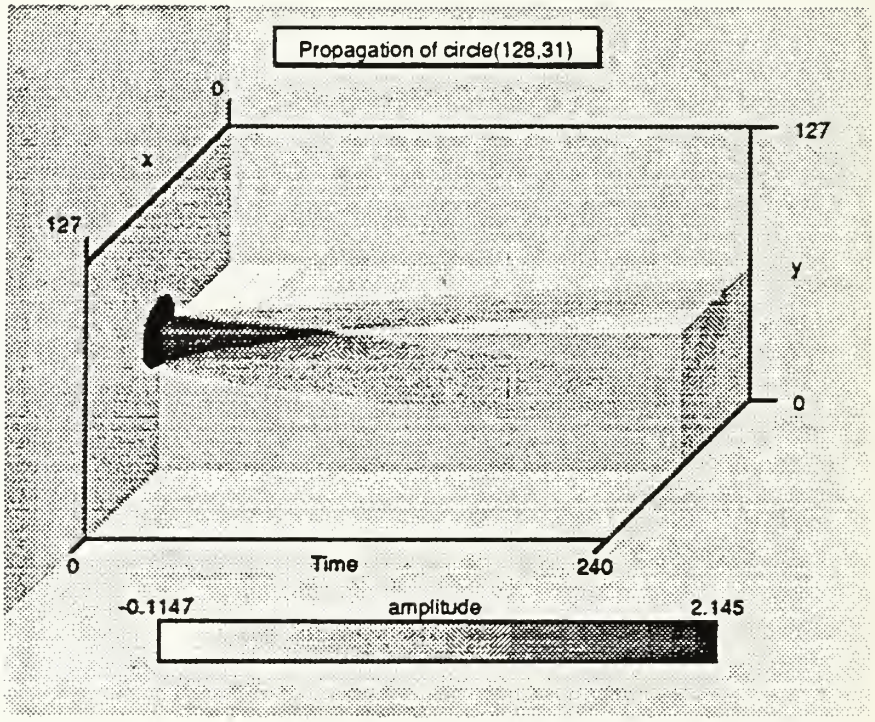


Figure E.5: *Circle* output for $d = 31$ samples. Dicer representation.

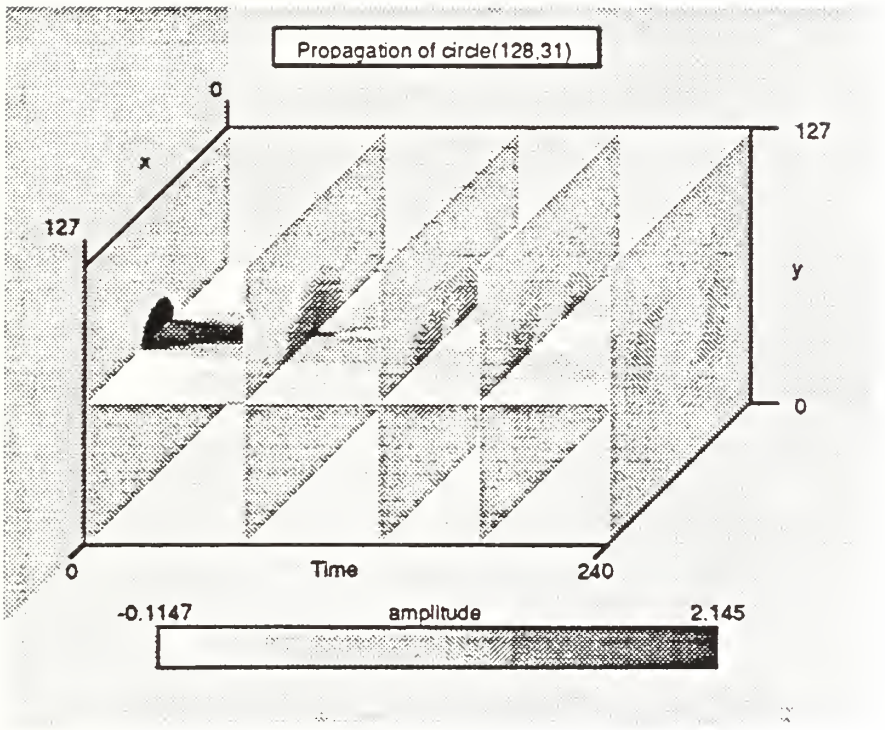


Figure E.6: *Circle* output for $d = 31$ samples. Alternative Dicer representation.

Bibliography

[1] J. Goodman, *Introduction to Fourier Optics*. New York: McGraw-Hill, 1968.

[2] D. Guyomar and J. Powers, "Transient fields radiated by curved surfaces — application to focusing," *J. Acoustical Society of America*, vol. 76, no. 5, pp. 1564–1572, 1984.

[3] D. Guyomar and J. Powers, "A Fourier approach to diffraction of pulsed ultrasonic waves in lossless media," in *Proceedings of the 1985 IEEE Ultrasonics Symposium*, (B. McAvoy, ed.), (New York), pp. 692–695, IEEE Press, 1985.

[4] D. Guyomar and J. P. Powers, "Boundary effects on transient radiation fields from vibrating surfaces," *J. Acoustical Society of America*, vol. 77, no. 3, pp. 907–915, 1985.

[5] D. Guyomar and J. Powers, "A Fourier approach to diffraction of pulsed ultrasonic waves in lossless media," *J. Acoustical Society of America*, vol. 82, no. 1, pp. 354–359, 1987.

[6] D. Guyomar and J. Powers, "A transfer function model for propagation in homogeneous media," in *International Symposium on Pattern Recognition and Acoustical Imaging*, (Bellingham, WA), pp. 253–257, SPIE Press, 1987.

[7] T. Merrill, *A transfer function approach to scalar wave propagation in lossy and lossless media*. Master's thesis, Naval Postgraduate School, Monterey, California, March 1987.

[8] J. Upton, *Microcomputer simulation of a Fourier approach to optical wave propagation*. Master's thesis, Naval Postgraduate School, Monterey, California, March 1992.

[9] W. R. Reid, *Microcomputer simulation of a Fourier approach to ultrasonic wave propagation*. Master's thesis, Naval Postgraduate School, Monterey, California, December 1992.

[10] P. R. Stepanishen, "Transient radiation from pistons in an infinite planar baffle," *J. Acoustical Society of America*, vol. 49, no. 5, pp. 1629–1637, 1971.

[11] P. R. Stepanishen, "Acoustic transients in the far-field of a baffled circular piston using the impulse response approach," *J. Of Sound and Vibration*, vol. 32, no. 3, pp. 295–310, 1974.

[12] P. R. Stepanishen, "Acoustic transients from planar axisymmetric vibrators using the impulse response method," *J. Acoustical Society of America*, vol. 70, no. 4, pp. 1176–1181, 1981.

[13] P. Stepanishen and G. Fisher, "Experimental verification of the impulse response method to evaluate transient acoustic fields," *J. Acoustical Society of America*, vol. 69, no. 6, pp. 1610–1627, 1981.

[14] G. Harris, "Review of transient field theory for a baffled planar transducer," *J. Acoustical Society of America*, vol. 70, no. 1, pp. 10–20, 1981.

[15] D. Guyomar and J. Powers, "Propagation of transient acoustic waves in lossy and lossless media," in *Acoustical Imaging, Volume 14*, (A. Berkhout, J. Ridder, and L. van der Wal, eds.), pp. 521–531, New York: Plenum Press, 1985.

[16] *MATLAB for MS-DOS Personal Computers, User's Guide.* The MATHWORKS, Inc., Natick, MA, 1990. User's guide.

[17] *AXUM, Technical Graphics and Data Analysis, User's Manual.* TriMetrix, Inc., Seattle, WA, 1989. User's manual.

[18] G. Tupholme, "Generation of acoustic pulses by baffled plane pistons," *Mathematika*, vol. 16, pp. 209–224, 1969.

[19] J. Lu and J. F. Greenleaf, "Experimental verification of nondiffracting X waves," *IEEE Trans. on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 39, no. 3, pp. 441–446, 1992.

DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Library, Code 52 Naval Postgraduate School Monterey, CA 93943-5002	2
3.	Department Chairman, Code EC Naval Postgraduate School Monterey, CA 93943-5004	1
4.	Director of Research Administration, Code 81 Naval Postgraduate School Monterey, CA 93943-5000	1
5.	Professor John Powers, Code EC/Po Naval Postgraduate School 833 Dyer Road, Room 437 Monterey, CA 93943-5121	5
6.	Professor Mathaias Fink Laboratoire Ondes et Acoustique ESPCI 10 rue Vauquetin 75005 Paris FRANCE	1
7.	Professor Hua Lee Department of Electrical & Computer Engineering University of California Santa Barbara CA 93106-9560	1
8.	Dr. Sidney Leeman Department of Medical Engineering & Physics Dulwich Hospital East Dulwich Grove London SE22-8PT United Kingdom	1